Analysis of Musical Structures: An Approach Utilising Monadic Parser Combinators

Alasdair J Anderson

This Thesis is submitted to the Department of Electronics and Electrical Engineering of the University of Glasgow in partial fulfilment of the requirements of the degree of Doctor of Philosophy

UNIVERSITY
of
GLASGOW

I declare that this thesis is a record of the original work carried out solely by myself in the Department of Electronic and Electrical Engineering at the University of Glasgow, during the period October 2001 to September 2005. Further rewriting was carried out during the period August 2006 to August 2008. The copyright of this thesis therefore belongs to the author under the terms of the United Kingdom Copyright acts. Due acknowledgement must always be made of the use of material contained in, or derived from, this thesis. The thesis has not been presented elsewhere in consideration for a higher degree.

Alasdair J Anderson

August 2008

Abstract

The work of this thesis seeks to further the use of computation in musical analysis. To a lesser extent it is hoped that it will provide some little evidence of a new angle on creating analytic elements through inference, and cast light onto some areas where analysis may be used anew.

Parsers for musical information are small in number, none have been implemented in functional languages, nor using monadic combination techniques. Few analytic systems are capable of, or even consider it necessary to, represent semantic ambiguity, and this is even more true of parsing systems. The work herein presented provides a system of unique monadic parsers built on combination that are capable of delivering several different types and depths of results.

Many computational-analytic systems are based on theories of similarity. The work presented here provides for analytic structures to be created through inference i.e. in the absence of known structures. This is believed to be the first instance of this type of structure generation in the field of music.

Acknowledgements

Thanks firstly to my perpetually enthusiastic supervisor Dr. Nick Bailey. Thanks also to all the members of the Centre for Music Technology without whom the last few years should have been very dull; Jered Bolton, Phil Kerr, Colin Law, Ji-Hyang Lee, Doug McGilvray, Niall Moody, Stuart Pullinger, Barry Short, Don Knox and Tom O'Hara. A general thanks to all those who have at some point asked me exactly what I was doing; in confusing you I often clarified things for myself. Perhaps most importantly, a debt of gratitude to all those educators who have increased my interest in music; Jim Bickel, John Madden and Noreen Silver in particular.

This research was funded by the Engineering and Physical Sciences Research Council.

Notes

Increasing volumes of information are available to researchers through the world-wide-web, sometimes to the detriment of the more traditional method of dissemination through printed material. Consequently a proportion of the references in this thesis include URL's where that may material may be found. Archival of material only available over the Internet is not yet a rigorous process and some material will eventually become lost. One possibility would be to include a data-image of all the web-only material cited in this thesis, though this has significant copyright implications, nor can there be a guarantee that any data-image, such as a CD-ROM, will still be retrievable in 10 or twenty years time. Should the reader be unable to locate a web-only resource cited in this thesis, they are referred to the WayBack Machine archive of web resources [2].

Contents

List of Figures

List of Tables

List of Code Samples

Chapter 1

Introduction

The aim of the work presented in this thesis is to advance the processes of musical analysis by exploiting and extending modern parsing techniques . Existing computer-based analytic systems have been tightly specified to particular genres, have required significant user interaction, or have been limited in the richness of the data with which they can operate. The work presented in this thesis seeks to avoid these shortcomings.

The methods presented in this thesis will not be bound to historical or partisan methodologies of scholarly music analysis. At the same time, these methods will address certain well-known problems in using computational systems for music analysis, such as detecting hidden and overlapping features, for which complete technical solutions do not yet exist.

Unique parsing methods will be presented and employed to produce analyses of musical works. Predication rules which discard possible information at intermediate stages will be avoided. Additionally, potential musical structures, not

discovered by the existing parser configuration, will be presented to the user as an integral part of the systems results, elucidating those areas where the system could be improved.

As well as performing typical analytical tasks more effectively, the system will allow new ideas and methodologies in music analysis to be implemented computationally. These should expand the current number and scope of uses for computer-based analytic systems, perhaps into new and unexploited fields.

1.1 Outline of the Thesis

As a guide to the reader, the outline content of each of the subsequent chapters is provided below:

Chapter 2 Investigates historical uses, methods and guiding principles of analysis and traces the evolution of various schools of analytic thought. Some short examples of representative analyses from the literature are provided for amplification.

Chapter 3 A discourse on how computation has been applied to musicology generally and to analysis specifically. A discussion of the appropriateness of applying more general computational techniques to music research.

Chapter 4 An introduction to Functional Programming and Parsing as well as the use of Monadic Combinators. An examination of the complicating factors in combining Music and Parsing theories is undertaken including the present state of the art. Context in an analytic sense,

parser implementations and suitability for music research are discussed.

Chapter 5 Presenting the initial data structures and simple parsers for them. Exhibiting the problem of overlapping elements and then presenting the novel solution to overcome this issue with respect to combination.

Chapter 6 Interfacing with a score editor to analyse examples from published musical work. Large scale combination, tokenisation, and the evolution of the data structure. A novel technique for inferential parsing is also introduced here, initially as a solution to a particular problem, but proving to be of significant use in its own right.

Chapter 7 Further evolution of the data structure and accompanying systems. Modifying temporal organisation. Altering the top-level combinator.

Chapter 8 Comparing system analyses, from various points during the software development, to human analyses, and a discussion of the differences and their significance.

Chapter 9 Reflections on the work as it stands, suggestions as to where it could be taken in the future, other fields in which the techniques might usefully be employed, and an estimation of the volume of work required to achieve these.

Chapters 2-4 comprise the bulk of reviewed literature considered relevant to the problems at hand. Chapters 5-7 describe the development and implementation of the software solution that grew out of the requirements from the literature. Chapters 8&9 provide results, reflections and outline some potential directions in which the work could be taken in the future.

1.2 Motivations for Musical Analysis

The desire to understand how and why musical sounds affect us is probably as old as our conception of music itself. For the ancients, music was one quarter of their *quadrivium* of philosophy [107, 35], and the pursuit of musical knowledge has abated little in the subsequent centuries. From the beginning of the Enlightenment, a considerable amount of scientific thought has been applied to the examination of the mechanical properties of *sound,* and this has continued into the Modern Era [50]. The understanding of the physical interaction of acoustic sources and environments has now become quite well understood [19]. Similarly, the perception of simple tones is quite well documented as a psychological process [45].

Sound is not always music though. Moreover what is heard to be music by one listener may be perceived as sound, or even noise by a subsequent listener. The work presented here addresses the least contentious end of this spectrum; organised (scored) sounds (notes), with relatively well defined acoustic properties, and yet we shall see that the variety of musicalities that can be encompassed is still wide.

1.3 Defining Music Analysis

Bent, writing as author [20] and contributor [97], describes analysis as

> *"... that part of the study of music that takes as its starting-point the music itself, rather than external factors. More formally, analysis may be said to include the interpretation of structures in music, together with their resolution into relatively simpler constituent*

elements, and the investigation of the relevant functions of those elements."

The *"external factors"* to which Bent refers, might include aspects of musicology, the social and historical compositional environment, a composer's commentary, or the acoustic performance environment. Common to all these factors is that they cannot be observed simply by looking at the score. That these externalities are not present on the page is the prime motivator for most analysis taking the score as the root data resource from which to draw its outcomes. It should also be understood that *"interpretation"* in this instance is not a translation into some other non-musical language, rather it may be read as a process of identifying the elements of the language through which a musical extract communicates.

There are also more abstract understandings of analysis. Theodore Adorno [9] offers an alternative view;

"... it is exactly in this direction that the way - the idea of analysis - really does lie: that is to say, composition understood as 'coherence', as a dynamic set of interrelationships. And it is within this set of interrelationships - if anywhere at all - that the meaning of the composition resides."

Adorno wishes to discover what aspects of the *zusammenhang*[1.1] cause a piece to hang together; what differentiates music from incoherent sounds. His view is more aesthetic than Bent's, but it is an aestheticism supported by observable fact.

Analysis then is the study of music and how it "works", whether this be at

[1.1] *Ger.* coherency or connectedness

a low, purely mechanical level, or at a higher level, where the mechanical blocks interact into a whole so internally complex that the listener ceases to observe the underpinnings, and begins to appreciate instead the overall impression of the piece.

1.4 Utilising Music Analysis

Music analysis has applications both in purely academic music research, and as a method of generating (or better, highlighting) data about a piece of music which is then used in some other system. The potential for making use of analysis in these areas is briefly discussed in Sections 1.4.1 to 1.4.3 on the following pages. Additionally presented, in Section 1.6 on page 12, are further areas where analysis might be fruitfully employed. Some of these opportunities are addressed in this thesis, and all have informed some part of the system design in various ways. In this way the utility of the system may be extended beyond its current scope without significant re-engineering.

1.4.1 Attribution

For historic works, determining the source of a particular piece of music, by reference to an existing corpus, has allowed works to be more accurately attributed to their original composers. For example, a number of works which had been assumed to have be written by Josquin[1.2], partly as a result of his prominence as a great Renaissance Composer, have more recently been re-attributed to lesser known composers (see *Josquin des Prez: WORKS - doubtful and mis-attributed works* in [97]). This confusion can easily arise when unscrupulous publishers, wishing to sell more manuscripts, would "discover" lost works of better known

[1.2] b nr Saint Quentin, c1450-55; d Condé-sur-l'Escaut, 27 Aug 1521

composers by re-attributing the compositions of lesser lights.

Analysis can be useful in chronologically positioning a particular work within a catalogue, if particular musical artefacts can be shown only to exist in a particular phase of the composer's work. One example is a work by Visconti[1.3], indicating that by its content that he may have lived longer (by some 10 years) than had previously been believed. This was inferred from the stylistic elements

> *"characterised by the galant manner, phrasing with triplets and a particularly frequent use of appoggiaturas"* [108]

that were revealed by analysis.

Many composers are professionals, and so there may be personal (financial) benefit in recycling thematic or structural material. It has been shown through analysis that some pieces are in fact drafts for later works, if a composer was dissatisfied with their less mature productions for example. Equally, an especially pleasing melody or structural feature may be employed in more than one published composition. Beethoven reused a theme from his *Cantata on the death of the Emperor Joseph II (woo87)* in the final act of the opera *Fidelio oder Die eheliche Liebe (op72),* also known as *Leonora.* These two works were written some 14 years apart, though contemporary listeners might have been hard pressed to identify the "original" material. The planned premiere of the *Cantata* had to be cancelled because the wind parts were found to be too taxing, and it was not performed during Beethoven's lifetime. Further examples of thematic reuse from Beethoven's work can be found in [104].

[1.3]b Cremona, 10 Jan 1683; d in or after 1713

For music composed more recently, there may still be outstanding copyrights, particularly in regions with strong music publishing legislation. The potential sums of money involved have brought some considerable weight behind a number of legal cases. The case of *Repp vs. Webber,* [39] in the USA, and others documented by the Columbia Law School *Arthur W. Diamond Law Library Musical Plagiarism Project*, show that often a considerable amount of musical analysis can be brought to bear in actions pertaining to millions of dollars in damages. Analysts will be invited to tender their expert opinions, using their skills to highlight either similarities or individualities in a manner amenable to the layman.

1.4.2 Segmentation

Many current areas of computer music, including some compositional techniques, archival/retrieval, and interactive learning, orbit around a common problem, namely the identification of musical structures. Considerable strides have been made in identifying notes from both acoustic sources [25] and written sources [23], essentially reducing the acoustic information to some form of meta-score. These are applications in music transcription, and the structural information they present must be explicit in the way it is delivered to the system (e.g. played by different instruments, or at different times); extrapolation is not a feature.

Techniques for identifying how notes might be organised, either perceptually by a listener or conceptually by a composer, are not yet well established and no best practice solution is accepted. The lack of standardisation is a good indicator that no current solution is sufficiently complete to be universally adopted. Extant systems frequently require either significant human pre-processing and

knowledge, or abstraction to a level where the human appreciation of music is almost totally ignored.

Gestalt theories of music, such as that of Lerdahl and Jackendoff [74], have produced rules for grouping notes based on the relationship between adjacent elements. Other models, such as that of Bod [24], are statistical and use training sets of pre-segmented musical works to form systems that will produce similar outcomes. The major shortcoming of this approach is that the corpora or works used as training sets are generally pre-selected on some common attribute such as style, composer or period. The variation of works in the training sets is correspondingly diminished which makes application to new corpora of limited, if any, use.

1.4.3 Archival and Retrieval

Music is a valuable data resource; many historical artefacts are now irreplaceable and many collections of written and recorded music exist throughout the world. Since recorded material can more efficiently be searched through pseudo-score meta-data than through the complex waveforms of the original sound recording, repositories should be aiming towards this as their primary query method (we would not expect, for example that a journal search would use facsimiles of the journal pages). Reports such as that of Bainbridge *et al.* [16], a panel of experts on various Digital Music Library implementations, indicate both the requirements for and the eagerness to find workable solutions. Some large collections, such as those of Indiana University [8], IRCAM [3], and John Hopkins University [4], have produced web-based interfaces to their archives. While the facility to present all this information to the world is laudable it seems to be more focused on delivering media electronically rather than using any musical

information as part of the retrieval process.

Query By Humming [22] systems are one popular avenue of research in archival/retrieval, though its error conditions (the degree of dislocation between the queries and the result "hits") are usually stochastic rather than based on specific musical rules. It is probable that this area will see the first commercial applications for retrieval software, though ultimately humming may not be seen as a suitable interface for serious musical research, and so it may be that new solutions will emerge from outside the research community. Commercial systems capable of identifying popular music from low quality (phone-call) sources are already viable (such as Shazam [7]), though the internal workings of these systems are confidential.

1.5 Computation in Music Research

The number of analyses that an analyst might be asked to produce may be quite high. Furthermore, whether used as a stand-alone process, or as a stage in a larger task, pen and paper analysis is a time consuming operation. Cook [36] posits that

> *"Analyzing a Beethoven symphony means living with it for a day or two"*

and this from an individual who is a professional in the field. Assuming that all stages (acquisition, analysis,representation) can be accelerated, it is reasonable to hope that this time could be reduced by a significant factor, reducing analysis times to hours or minutes. An ultimate goal would be to reach a stage where an analysis can be produced in less time than it takes to perform the piece i.e.

better than real time, which even an expert listener could not hope to achieve. Having to manually perform even a single stage of the process could increase the time taken by a significant factor.

Considering that; the number of available digital representations of music increases continually, that for some composers composition may completely bypass pen & paper, and that systems to acquire musical data from both acoustic and visual sources are highly developed, such a conversion should become increasingly less necessary. Even in a case where the score must be entered to a system manually, it will be a one time operation, after which multiple analytic and analysis-reliant applications can be performed.

Computational power can provide an improvement in the amount of time taken to perform even a basic level of analysis. The use of logical routines will also remove human error and subjectivity from any analysis performed. Increases in computational power, with Tera-Flops of processing power available to the domestic user, can only make computation a more effective catalyst in any analytic procedure. Initial reviews of the literature also indicated a number of unproductive approaches; simple pattern matching routines, incomplete pitch representation, systems that require a very high level of computer or music theory facility. The research presented in this thesis began not exactly with a clean slate, but certainly without prejudice about what should be the right way forward. The system should be extensible beyond what was originally envisaged, thus care was exercised throughout so as not to rule out any future avenues for expansion.

1.6 Further Avenues of Research

There are a number of undeveloped areas in the application of analysis. Some of these are only partially exploited, and others may be entirely novel. Below are two examples of arenas where analytical tools may be used as secondary aides to an alternate primary purpose.

1.6.1 Performance Variables

While there is some literature on analysis as a means for informing performance, such as that of Berry [21], it often tends toward broad generality in its principles, and tight specificity in its application. The inclusion of a small number of analyses showing what the performer should be making clear is not backed up with any suggestion as to how this might be done.

It is made clear however that creating performances which reinforce the musicality of a composition requires an execution structure that reinforces the structures of the work at hand. Execution structures are naturally instrument specific, but should encompass most variables open to the performer, such as articulation, fingering and other aspects of technique.

Unfortunately, available fingering systems, of which Sayegh [98] is typical, view fingering as a problem of simple minimisation, and are certainly devoid of style specific performance elements. A performer might also consider aspects such as the ease of execution, with respect to their expertise or facility with the instrument, as having an impact on the performance as an audience will perceive it. Automated performance systems should also benefit from musically relevant articulation patterns, to soften the harsh edges often produced by exactly quan-

tised performances, and there is some published work in this field such as that of Clynes [34], based principally on stochastics.

Parncutt has produced more recent work [87, 33], comparing a model developed from practical ergonomics and instructional texts with actual fingerings of performance pianists. From the evidence of these papers it does not seem that this has successfully been extended to a complete work of music, and there is no detail of a computational implementation.

1.6.2 Style Checking

Compositional systems based on genetic algorithms [82], neural networks [53] or other artificial intelligence techniques require some fitness function to select which branch to evolve to the next generation. Where some composers, such as Xenakis, have used partially algorithmic or stochastic techniques and *chosen* the most pleasing outcome, fully automatic systems require a methodology for ranking various outcomes, if the musical output is not to seem too overtly engineered.

Computerised composition based on rule sets to produce music of a prescribed style have existed for nearly half a century since the early IBM computers became available to the academic community [54], but such systems are felt by some music professionals to produce less than perfect results. It may be the case that the results are in fact too perfect because no rules are stretched or broken. The listener's cycle of expectation and resolution is never challenged, resulting in a lack of musical engagement. If a system begins with the set of all possible pieces, which are filtered down with some set of analytic rules, the outcome will most likely differ from that produced by a purely generative approach.

1.7 Parsing and Functional Programming

It was considered at the outset that parsing could be a suitable method for representing the necessary levels of ambiguity that would be inherent from music. Parsing theory is in a constant state of advancement, and though some attempts have been made to apply then-current parser techniques to aspects of music research, they have largely fallen by the wayside as possible avenues of research.

Functional languages (Section 4.1 on page 47) have an easy to follow syntactic structure, and are more intuitive to the novice computer user than imperative languages. This should prove to be important, as in order to become a useful tool to the community, any software should be accessible, both in terms of use and extension, to as many users and researchers as possible. Haskell is one of the better documented functional languages, and has an active user-base[1.4].

1.8 Direction of Analysis

The reader may recall that Bent [20] was describing a "top down" process, which might take as a first step (top-most level) a superficial description of the piece as a whole, for example, being in the form of a sonata, symphony or concerto. The next step would be to identify elements such as the exposition or development, the next to identify thematic information and so on. It should be possible by contrast to consider that the problem can be approached from the bottom up, from the level of the *"simpler constituent elements"*. These would then combine and interact to provide the higher levels of structure. Looking "up" may reveal

[1.4] The haskell-main and haskell-cafe mailing lists provided through *www.haskell.org* provide a valuable resource of code examples, both in terms of utility and instruction

multiple structures formed from the constituent micro-elements, and the function of the elements may be different in each of those macro-structures.

What advantages could be gained by navigating upwards through the structure? It is precisely the multitudinous opportunities that will surely become apparent as a result of not being bound as a first step to identifying a piece as being one thing or another, judging a book, as it were, by its cover.

> *"The 'musical' is any sonorous fact constructed, organised or thought by a culture"*[86]

Listeners or analysts, may not yet be aware of the form in which a newly discovered work is shaped. On the contrary, it may be easier to sketch out new forms based on upward looking analysis.

1.9 Terminology in Music and Science

It is the perception of the author that truly integrated research in music using computers does not yet exist; real expertise is usually only expressed in one field or the other (Computing OR Music). Subsequent to this, there is perhaps a misnomer at the centre of the application of the sciences to theories of music, and it hinges upon the use of the word "theory". In science, a theory is a proposition, grounded upon knowledge, and which may be held to be true until proven to be false. Music theory is quite different, with every rule having circumstances in which it never holds true; theory is the collection of rules and there is no requirement for co-dependence.

It is certainly possible to approach music in a rigorous rule-based way, but

rather than wrap music theory into strict boolean conditions, why should research not be aiming to adapt computation to better handle the ambiguous properties of music. Little consideration has been given in the literature as to whether "rules" of existing analytic systems are, or were ever intended to be, general cases, or whether they either hold for some forgotten constraint, or are specific implementations of more general theories. Schoenberg distinguished his own theories from scientific theory by saying

> *"whenever I theorise, it is less important whether these theories be right than whether they be useful as comparisons to clarify the object and to the study perspective"[99]*

What seems certain is that the majority of human decisions in music analysis are subjective, and based on context.

Chapter 2

Theories of Music Analysis

This chapter gives an overview of the historical development of pre-computational analysis of music. This should provide the reader both with a broad outline of the development of musical analysis, but also draw out the salient developments in terms of the systems described later in the thesis.

Part of the intrigue of music is that different listeners extract different information from the same musical work. Equally, a single listener may hear different facets of a single piece at each listening. Which, if any, of these listening experiences is the "correct" one? Should the correct experience necessarily coincide with the intentions of the composer? Analysis may help to answer these questions, but only if there can be some certainty that any methods used are capable of at least considering a maximal number of these "hearings". From this maximal number, how far should an analysis seek to condense or summarise it's findings?

A structure has both a form and a function, and the elucidation of structure is dependent on their symbiotic relationship. A function of a structure at any

given hierarchical level is a product of its form with respect to other structures at the level. Conversely, a structures form is only valid provided it performs a specific function at some contextual level; form without function is not valid form. Theorists have focused on different sets of functionality which then infer different forms within a piece of music.

2.1 The Evolution of Analytic Theories

The precise function which analysts expect a form to fulfil is dependent on the philosophy that motivates their research. An analysis focusing on our perception of music (or at least how science currently perceives our perception, if that is not too much of a tautology) must show what perceptual properties are exploited within the composition. In contrast, an analysis looking at compositional process might seek to show the evolution of the inspirational spark or to discriminate the essential musical content from the inessential, thus highlighting the compositional form.

To some extent, theories and processes of analysis are created to explain, or even justify, new compositional styles. Particularly when the aesthetics of the new style are at odds with those of the style it supersedes, elucidation of process often helps qualify (intellectualise) the artistic merit of that new form. Even as new nomenclature forms have been created to instruct the performers, new ways must be found either to relate these to the existing analytic methods or to create entirely new methodologies of analysis. Over time therefore, various systems of analysis have evolved, even as music itself has evolved, and with them, ideas about what music means to us as listeners, and how we create music both conceptually and physically.

2.2 Time-Line of Analysis

Cook [36] suggests that there a two distinct historical phases of musical study, and that

> *"... until some two hundred years ago such speculations bore little affinity to what we nowadays mean by the term 'musical analysis'."*

Bent [20] places the establishment of analysis almost 100 years later in the late 19th Century but effectively describes three periods (of which only the third encompasses analysis as he sees it);

- C11 - 1750
- 1750 - Late C19
- Late C19 - present

Many historically useful techniques are deemed inessential today, having been superseded and often decried by subsequent methods and their progenitors. Their impact on later treatises is essential in as much as it informed their construction, that is to say that none of these subsequent methodologies have been developed in isolation.

2.3 C11 to 1750

In its infancy, analysis was concerned with extracting musical generalities, common to music as a whole, rather than those features that made a specific work unique. It was only during the early C17 that real consideration was given to

the school-book triumvirate of Melody, Harmony and Rhythm. During the preceding 6 Centuries then, analysis was mostly a cataloguing process, describing broad generalities such as the mode in which it was written, or the part of the religious service it was intended to accompany.

This sort of simple analysis would not merit a computational approach, as most composers have clearly declared this sort of information (E minor Violin Concerto) in titling their work. It is useful information to have as meta-data within a score file, and can in most cases be assumed to be accurate without further investigation when acquired from a printed score. Then, principally during the C16, but earlier in some isolated circumstances, the (re)discovery of Classical (Greco-Roman) concepts of rhetoric pushed theorists towards establishing formal organisation within works [75].The first analyses, as we might term them, to be laid out are said to be those by Joachim Burmeister[2.1] in his Musica Poetica [27]. This work also includes one of the earliest written description of analysis

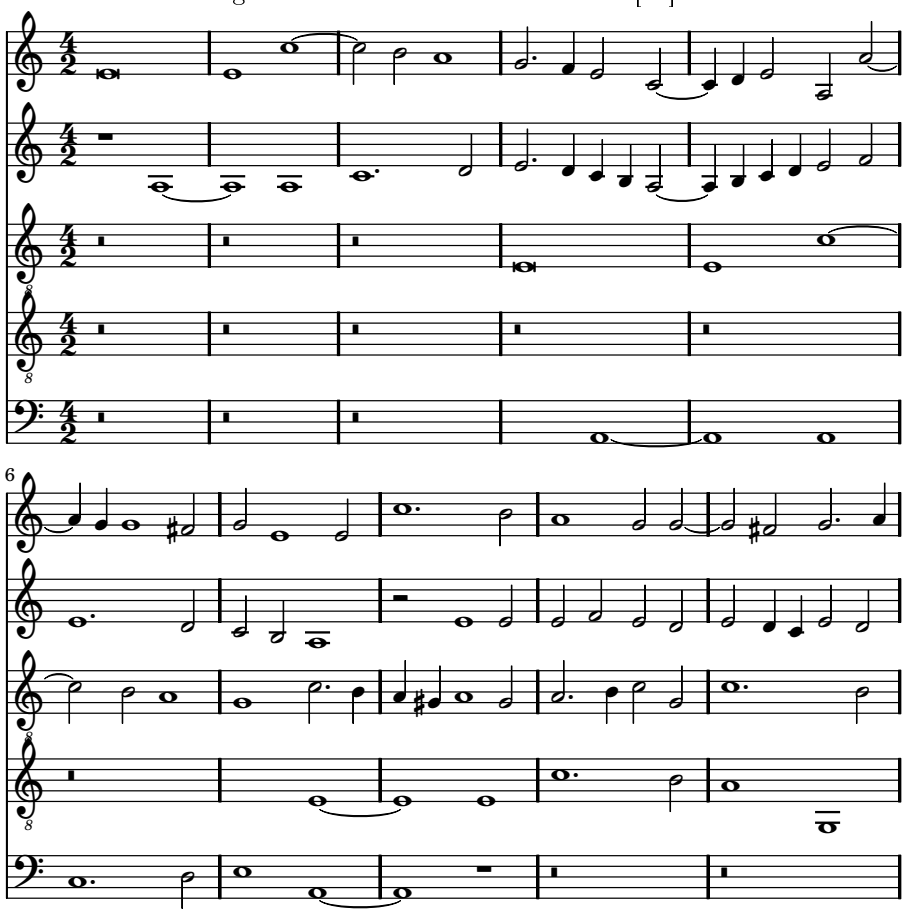
> *"The piece is to be divided into its affections or periods, so that the artfulness with which each period takes shape can be studied [and adopted for imitation]. There are five areas of an analysis: 1) investigation of mode; 2) investigation of the melodic genus; 3) investigation of the type of polyphony; 4) consideration of the quality; 5) sectioning of the piece into affections or periods."*[27]

Burmeister refers to this section of his work as *De Analysi sive Dispositione*, which could be translated as *Of Analysis and hence Of Arrangement*; he seeks to elaborate the arrangement of elements through his analysis. A number of analyses are included in the text, though all are verbal descriptions accompanying the score. It is not clear whether the absence of annotated analyses was

[2.1] b Lüneburg, 1564; d Rostock, 5 March 1629

a function of wishing to *"spur his readers to transcribe the music themselves"* (Riviera & Ruhnke [97]), or of the technical difficulty and cost of including such features in printed material. In the extract contained in Figure 2.1, Burmeister perceives the presence of both *fugis realis*[2.2] (such as between parts 1 and 3) and *hypallage*[2.3] (such as between parts 1 and 2).

Figure 2.1: Lasso *In me transierunt* [27]



[2.2] voice by voice imitation using identical or similar intervals.

[2.3] voice by voice imitation using inverted or complementary intervals e.g. major 3rd/minor 6, perfect 5th/perfect 4th

The rhetorical musical figures (26 all told) his analyses reveal are very much a function of a particular compositional style, prevalent at the time of writing, which was moving towards the in-temporal harmonic structures created in polyphonic writing. Regardless of the merit of these analyses, it is clear that understanding and conveying the analysis is greatly hindered by the medium of delivery (textual description). This media can also introduce its own layer of ambiguity as the reader is forced to reconcile the two representations of the work: score and analysis.

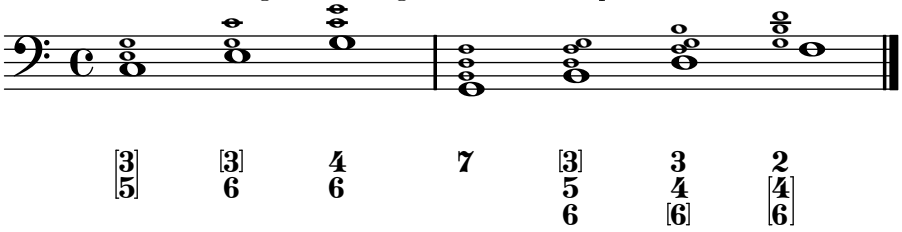
In considering analysis of compositions from this period, it should be remembered that a large part of the musical interest in a piece was, at this time, added at the point of performance. Treatises on the realisation of Figured Bass and the art of Embellishment may well be considered influential on analysis methods in as much as later works will explicitly state the desired realisations and embellishments.

Figured Bass [2.4] is a short hand notation used to indicate the structure of chords to be executed over a notated bass line. As a widely understood harmonic shorthand, it would be a logical tool for annotating analyses at this time, and has a low overhead for typesetting. A work could be reduced to bass and figures by a teacher so that a pupil might rework the piece. Comparing the study to the original would highlight how a *maestro* had dealt with a particularly taxing problem. By the same token, having a pupil reduce a master work could elucidate what structures, in terms of figured bass progressions, were to be found in masterful compositions. Furthermore, a publisher could reduce the size and number of paper parts required to outfit an an ensemble, using this (harmonically) lossless encoding. Examples of simple notation can be seen in

[2.4] (*bezifferter bass [Ger.], basse chiffrée [Fr.], basso numerato [It.]*)

Figure 2.2, though the bracketed digits would normally be omitted as they are implied by the unbracketed digits.

Figure 2.2: Figured Bass Examples



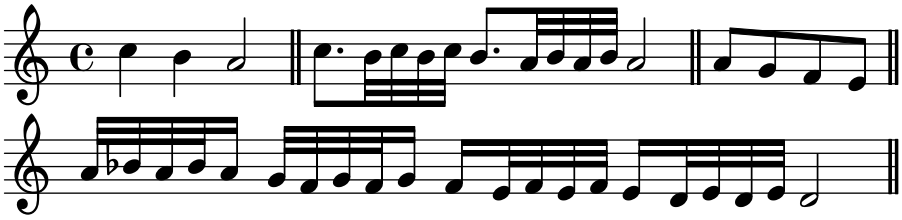
While this sort of score encoding might have been beneficial to computational music analysis in previous decades, present day data densities should encourage a maximal encoding of the score. To give some metric to this; a Denemo representation of a Bach cello suite would be under 100kB, and a pdf image need not be much larger, compared to a CD quality audio file of the same work at around 200MB (performance times by different cellists vary by several minutes).

Embellishment is, as one might expect, the art of extending a simple structure by the addition of superficial micro-structures at the point of performance. Brown [26] describes two forms of embellishment which existed during the sixteenth century: Graces and *Passagi*[2.5]. Four of the graces he describes (*tremolo, groppo*[2.6]*, clamatione*[2.7] and *accento)* are taken from Diruta's discursive manual on organ technique [47]. Diruta describes his *passagi* as *minuta*[2.8], and they expand common melodic intervals using melodic formulae. Another contemporary writer, Ganassi, provided an exhaustive 175 separate ornamentations of a single cadence [46]. Figure 2.3 on the following page shows two examples of *tremoli*

[2.5] *[It] passage*
[2.6] *[It] A cadential upper-note trill, often with a turn at the end*
[2.7] *[It] A Portamento up to the first note of a passage from a 3rd or 4th below*
[2.8] It: rough translation

from Diruta.

Figure 2.3: Tremoli Example [47]



Later analysts (2.5.3) would reverse this process, considering these embellishments to be inessential.The advantage of a notation for embellishment (which lives in figures such as the turn and mordent), is that they allow both the elaborate and simple forms of a piece to exist on a single score. These glyphs do not individually represent ambiguity *per se*, but the short-hand they represent is certainly a model for notating multiple performance instances.

Jean-Philippe Rameau[2.9] was perhaps the most influential of 18th Century music theorists, but less for his treatment of large scale structure than for his investigation of musical physics. Most of the material that is relevant to the present discourse in his *Traite de L'Harmonie Reduite a ses Principes Naturels* [89] can be found in the second volume, though interesting mathematical and geometrical observations are to be found throughout all four volumes. As an *ouevre* it may be considered to be analogous, at least in its attempts to reduce music to a deductive science, to Newtons *Philosophiae Naturalis Principia Mathematica* and other deductive works of the period. Rameau was to a great extant retreading the work of the Pythagoreans in looking at musical relationships in terms of the ratio of lengths of vibrating strings , reducing the diaspora of figured bass into his theory of *basse fondamentale* (implied bass).

[2.9]b Dijon, bap. 25 Sept 1683; d Paris, 12 Sept 1764

2.4 1750 to Late C19

Joseph Riepel[2.10] produced works [91, 92, 93] of a didactic nature, adopting the master-pupil dialogue style used by the ancient Greeks. His works focus on combinatorial application of musical factors to produce multiple arrangements of initial material, but rely somewhat heavily on each measure being a structure in itself, and these structures being quite portable in terms of their ease of re-use. This is only a useful segmentation where the rhythmic structure is both simple and regular, which is increasingly an assumption that cannot be justified for subsequent compositions.

Johann Philipp Kirnberger[2.11] was established as the most influential member of a group of Berlin-based theorists, including C. P. E. Bach. His published works [68] were of a practical, didactic type, partly as a response to the comparative lack of instructional material left by J. S. Bach, in contrast to his musical output. Whilst building on the species-counterpoint systems of Fux and chord treatments of Rameau, he retained the bass as the primary motive in his compositional theories.

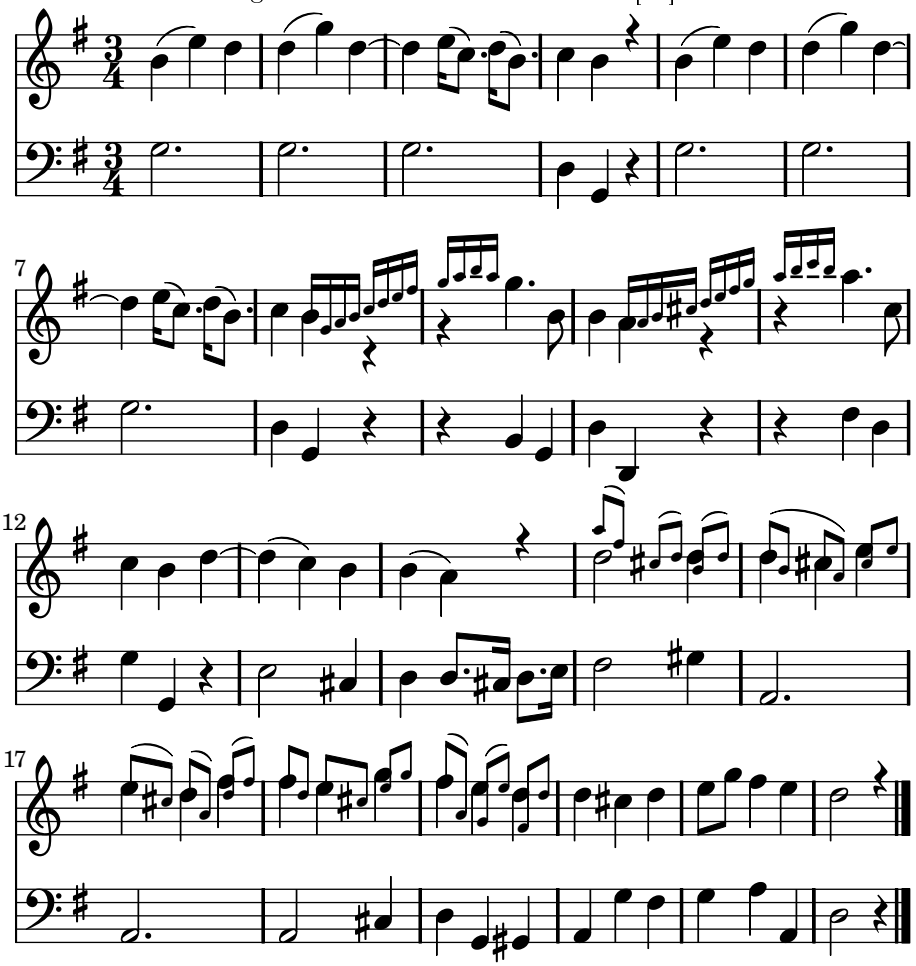
Heinrich Christopher Koch[2.12] built upon the theoretical works of Kirnberger and Riepel, as well as the more general art writings of Johann Sulzer. Koch's work [69] bridges the competing theories regarding the musical supremacy of harmony or melody. He declared the key or mode to be the *urstoff der musik*[2.13], as well as examining music on both a micro and macro-structural scale. From the English translation [17] of *Versuch einer Anleitung zur Composition*, Koch describes the process of looking at the construction of melody thus

[2.10] b Deutsch-Hörschlag, Upper Austria, 22 Jan 1709; d Regensburg, 23 Oct 1782
[2.11] b Saalfeld, bap. 24 April 1721; d Berlin, 26 or 27 July 1783
[2.12] b Rudolstadt, 10 Oct 1749; d Rudolstadt, 19 March 1816
[2.13] *Ger: original material for the music*

> *"we must separate its structure into individual sections; from the manner of their connection and from their external characteristics, we must try to abstract the rules and maxims which were the basis for their original combination"*

Breaking down the compositional process into three parts, *Anlage - Ausführung - Ausarbeitung,* roughly translating as creation - construction - elaboration, was a further step towards analysing music from inception forward, hinting at generative techniques. Another of Koch's novel (and since, much copied) ideas was an attempt to describe musical structure in grammatical terms; phrase, clause and sentence. Within this terminology, phrases were formed from complete and incomplete segments or *einschnitt*, and phrases could either be termed opening (*absatz*) or closing (*schluss-satz*). *Versuch einer Anleitung zur Composition* also contains a reductionist analysis of the second aria from *Tod Jesu* by C H Graun, which can be seen in Figure 2.4 on the next page.

Figure 2.4: Reduction of *Tod Jesu* [69]



Antoine Reicha[2.14] worked in Paris and brought theories comparable to those of Koch into Francophone music circles. His work is characterised by extensive use of other composers work in his examples. In fact he would remark that

> *"It is with music as with geometry: in the former it is necessary to prove everything by music examples, just as it is with the latter by*

[2.14] b Prague, 26 Feb 1770; d Paris, 28 May 1836

geometric figures"

Jerome-Joseph Momigny[2.15] based his structural analyses on the principle that all music must move from upbeat (*levé*) to downbeat (*frappé*), and never from down to up [43]. The smallest unit of musical structure was then the *proposition musicale,* consisting of consecutive up and down beats. Momigny also introduced a concept of a 27 note tonal space (7 diatonic tones plus 5 instances of the chromatic alterations ♭, ♯, double-♭ and double-♯).

Carl Czerny[2.16], in his School of Practical Composition [40], was the first to produce a stand-alone treatise on instrumentation and form, without including the previously obligatory exposition of form and counterpoint. This *schule*, the fourth and final volume of his instructional masterpieces to be assigned a centurion Opus number (600), reworks some ideas from the earlier *Fantasie-schulen.* Czerny espoused that

> *"the composition must... belong to a species already in existence; consequently, in this respect, no originality is, in general, necessary"* [20]

which would contrast markedly with the idea of the artist as an original genius in Romantic philosophy.

2.5 Late C19 to Present Day

The last 150 years of analytic and more general music theory thinking have produced a number of distinct strands. Some are motivated by philosophy, such

[2.15] b Philippeville, Namur, 20 Jan 1762; d Charenton, 25 Aug 1842
[2.16] b Vienna, 21 Feb 1791; d Vienna, 15 July 1857

as Gestaltism or Semiotics, some are created specifically to accommodate radical directions in composition, such as that of the second Viennese School.

2.5.1 Gestalt-based Theories

Gestalt[2.17] psychology emerged from the philosophical works of Goethe, Kant, & Mach. It is based on the premise that the human mind has deeply embedded principles of organisation, particularly in terms of macro-level events, rather than constituent events. In terms of visual cognition for example, it is said that objects that are spatially, geometrically or spectrally close, are likely to be grouped together. The popularity of Gestalt thinking in the closing years of the 19th Century would see its principles being applied to many fields of human perception.

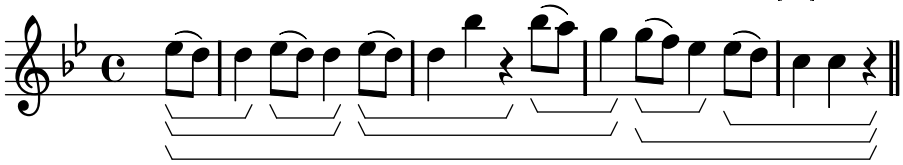
The first psychologist to apply Gestalt principles to music was Christian von Ehrenfels[2.18]. In 1890 he observed that a melody does not lose its melodic identity under transposition, despite every note having changed, though this is only strictly true for music performed in equal temperament. Another interesting observation he made was the concept of figure-ground differentiation, whereby only a subset of the information available (the figure) is passed to the brain for identification, and the rest is left to lie (the ground). Interestingly, this terminology is the reverse of the nomenclature used by Czerny, whereby the figure was the inessential surface detail, and the ground the underlying musical process.

The work of Lerdahl and Jackendoff [74] has become a touchstone for gestalt-based theories in the last twenty years. The rule-based approach lends itself to systematic analyses such as in Cambouropoulos [29], who cites experimental

[2.17] Ge. meaning form, figure or shape
[2.18] b Rodaun near Vienna,June 2 1856; d Lichtenau, September 8 1932

support for the theories. The Mozart example in Figure 2.5 is produced as a result of simple grouping preference rules (GPR's) based on differences in pitch, duration, onset, dynamic and articulation.

Figure 2.5: Lerdahl and Jackendoff GPR Analysis of Mozart [74]



In the work of Lerdahl and Jackendoff, such an outcome is a non-singular intermediary point and further rules are introduced to produce a final analysis.

2.5.2 Semiotic-based Theories

In the sense in which analysts have used the term, semiology[2.19] was coined concurrently by Saussure [44] and Peirce[88], though both were retreading an idea first put forward by Locke[76], which concerns the study of signs, or systems of signs.

> *"There are strong arguments that music inhabits a semiological realm which, on both onto-genetic and phylogenetic levels, has developmental priority over verbal language."* [84]

To analyse music semiotically is to decompose a piece into significant parts and to investigate the interactions and relationships of these parts. Because the dis-

[2.19] Semiotics can be said generally to fall into 3 areas:

- Semantics: How signs relate to the things they denote
- Syntactics: Within formalised structures, how signs relate to each other
- Pragmatics: The relationship between the sign itself and he impact of that sign on its user.

tribution of a particular sign is of import, semiotic analysis is sometimes referred to as Distributional Analysis.

Nattiez [86] breaks his semiotic analysis of music into two parts; Paradigmatic and Syntagmatic. Analysis begins at the *neutral* level where

> *"... one does not decide a priori whether the results generated by a specific analytic proceeding are relevant from the poietic or aesthetic point of view."*

Ruwet [95], first published in 1966 [94], puts forward the case for relationships between musical signs being as a function of recurrence or repetition. Tarasti [105], in his examination of Wagner's *Siegfried*, shows the intervallic structure of the sword leitmotif (Figure 2.6), maintained in the passage in which Siegfried forges the sword (Figure 2.7 on the following page) and other locations throughout the opera.

Figure 2.6: Sword Leitmotif [105]

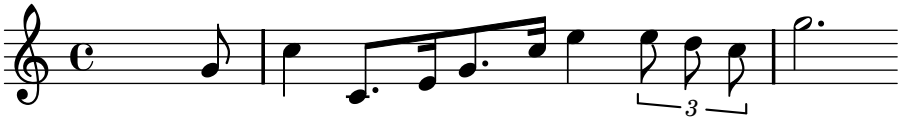
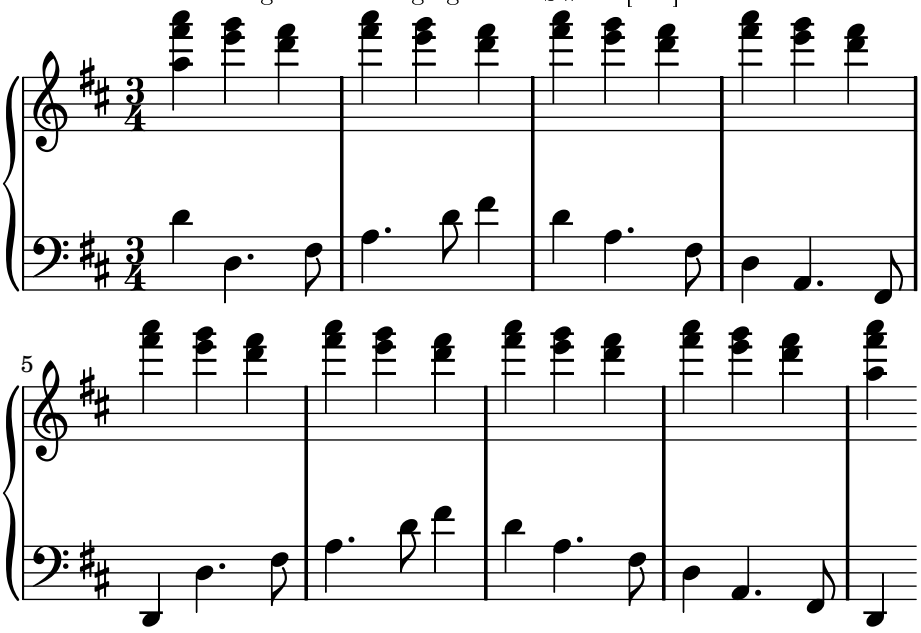


Figure 2.7: Forging of the Sword [105]



2.5.3 Schenkerian Analysis

The works of Heinrich Schenker[2.20] have inspired a broad church of analysts to follow, build, and fork away from in their own work. Schenker's basic tenet is that the fundamental structural form of music is the triad, both in in-temporal and arpeggiated forms. These are elaborated through a variety of techniques to form a piece of music. Schenker's is essentially a reductionist theory, that the musical whole can be reduced to a genesis cadence, and circumvents some of the pitfalls he perceived in Roman numeral notation.

His work is both sufficiently modern and widely distributed enough to have become partially synonymous with the process of analysis. It also spawned a

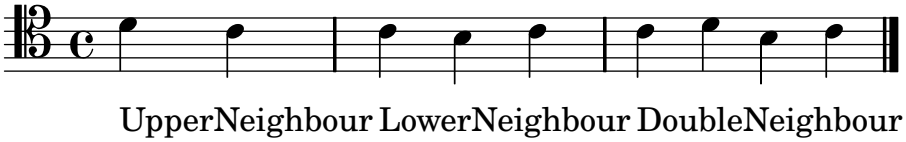
[2.20] b Wisniowczyki, Galicia, 19 June 1868; d Vienna, 13 Jun 1935

neo-Schenkerian school, expanding the scope of the technique by applying it to works from periods which Schenker himself deemed either unsuitable or without merit.

Schenker's lowest level elaborative structures are concerned with the relationship between notes. Structures, or rather, diminishing processes, include the neighbour and passing notes, the consonant skip and arpeggiation. The first two are fully described here as they are employed in later chapters of the thesis, while a short description of the remainder is also provided. For a more complete examination of Schenkerian analysis may be found in Forte [48].

A basic *neighbour* note motif (Figure 2.8) starts on a pitch, moves away no more than one diatonic step and returns to the original pitch.

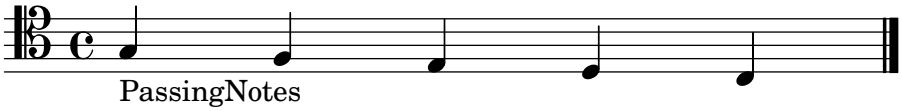
Figure 2.8: Neighbour Note Instances



More elaborately, a fourth note may be inserted before the return to the starting pitch, but with the diatonic step opposed to that between the first and second notes. Turns, mordents and trills can all be expressed in terms of the neighbour note diminution. More generally, the neighbour note is used to augment a single note, adding motivic interest to a held pitch.

The *passing* note motif (Figure 2.9 on the following page) is also built from the diatonic step.

Figure 2.9: Passing Note Instances



The number of passing notes is not limited, but the diatonic steps should continue in the same direction. Any part of an ascending or descending scale can be expressed as a number of passing notes, and the general case is of adding motivic interest to a specific interval. In figure 2.9, a perfect fifth is augmented with passing notes.

A leap of more than one diatonic step is termed a *consonant skip*. The *arpeggiation* structure consists of two or more consonant-skips which as a whole can be considered to form a harmonic chord, though this will to some extent be constrained by the limits of what is harmonious in context. This makes arpeggiation somewhat difficult to define rigorously.

2.5.4 Analysis of the Second Viennese School

Arnold Schoenberg[2.21] was the major driving force in the Second Viennese School, as well as being a respected critic of the music that had gone before him. Since this thesis deals exclusively with tonal music as its subject for analysis, the later techniques developed for and by the serialists are not discussed. Schoenberg's ideas though are certainly of interest, particularly his thoughts on coherency.

> *"A strip of paper is glued to a piece of cork; a chicken feather is glued to this, to which a nail is tied with string. Such things*

[2.21] b Vienna, 13 Sept 1874; d Los Angeles, 13 July 1951

are certainly connected. But the whole is lacking any characteristic relationship. Likewise a line merely tied to a stick can form a fishing rod. Here it is the sense, the purpose, that legitimises the connection of these objects" [100]

Schoenberg's move to the USA in the mid-30's exposed many new students to his work, and helped to to open the schism that can be often be seen between European and American analysts.

2.5.5 Functional Analysis

The term *Functional Analysis* was introduced by Hans Keller[2.22] in a 1956 article [66], but was preceded by two published analyses, both of works by Mozart, in that same year. This functional analysis is unrelated to the work of Hugo Riemann[2.23], which deals with analysing the function of chords in a progression. Attempting to formalise his burgeoning theories [67], Keller's key themes are *wordlessness* and *simplicity*. The full impact of this wordlessness would surely have been felt during the September broadcast of Keller's analytic score (for string quartet) for Mozart's D minor Quartet K.421. Its rationale rests on the principle that writing about an analysis cannot convey any information about the piece that the listener is not already cognisant of.

...either you know that it's the second subject or you don't. If you do, you don't need to be told; if - despite the fact that the whole movement has just been played to you - you don't, the pronouncement ["And here is the second subject"] still remains more of a distraction than a piece of relevant information...

[2.22] b Vienna, 11 March 1919; d London, 6 Nov 1985
[2.23] b Gross-Mehlra, nr Sondershausen, 18 July 1849; d Leipzig, 10 July 1919

Perhaps his greatest claim is to the removal of "subjectology", in as much as written terms such as "second subject" are entirely eliminated, replaced by a direct musical quotation.

2.6 Summary

The first conclusion we can draw is that an analysis should be encapsulated as far as possible and practical within the score itself. While each picture may not be truly worth a thousand words, the meaning is certainly more tightly conveyed when graphically juxtaposed.

We can then note the development of a limited number of glyphs capable of encoding both harmonic and melodic content, whose precise execution is not proscribed. It can be convincingly argued that the musical meaning remains unchanged, though this is inevitably due to their extant uses, and they could certainly be extended or re-envisioned to encapsulate the possibility for different meaning.

This overview gives us a footing, a minimum capability as it were, for a computational analytic system. It is true that we do not want the system to be bound to any one of these individual methodologies, but equally, a system which cannot encompass even one of these methodologies leaves itself open to criticism. Considering end use scenarios in the construction of a software system is essential, but as we shall see, the system built on Monadic Parser Combinators will allow us even more flexibility.

Each of the historic methods has one aspect in common; a unitary represen-

tation of the analysis, a solitary conclusion. It could be argued that the paper media in which they are disseminated forces the analysts to opt for the most satisfactory single outcome, but it seems more reasonable to state that the methods themselves have a unitary result as a goal. It is common for parsers also to be unitary in their output, though as we shall see, the Monadic Parser Combinators allow to step away from the usual parsing paradigm to allow us much more complex and complete descriptions.

Chapter 3

Computation in Music Research

Music research has made use of computational power almost since it became available to the academic community during the 1950's. To this extent, a great deal of consideration has been given to aspects of music as they pertain to computation, not only in the area of analysis. Reviewing the issues that have been encountered in computational music research, and the solutions presented to date, is invaluable in designing any comparable system such as that presented in this thesis.

Many of the issues occur in more than one field of music research, and so a discussion both of fields of research, and of individual issues, is necessary to provide a complete overview of both issues and solutions.

Music Information Retrieval (MIR) became popular as a terminology around

the year 2000. This was the year that the annual *International Symposium on Music Information Retrieval* (ISMIR) was first hosted, though prior to this MIR had also been used to describe *Multimedia Information Retrieval* as part of the ACM *Special Interest Group on Information Retrieval* (SIGIR).

Any system which creates new data about a piece of music can be said to have retrieved information, though most MIR systems are aimed at answering specific queries such as

How many time does this phrase appear in that piece?

or

Which piece from our library contains this phrase?

These queries require a degree of *a priori* knowledge if results are to be indicative. It is only possible to ask the question *"How many times does this phrase appear in that piece?"* if it is known that the phrase appears at all. Thus it seems that many MIR systems are in fact corroborating, confirming or quantifying some supplied statement that the user believes to be true.

It would appear from the literature that there are a smaller number of systems aimed at telling us something about a piece without any prompting, Music Information Generators (MIG) as it were. MIG systems must first establish what are appropriate candidate queries before performing a MIR process to assess the fitness of these candidates.

Common to both MIR and MIG systems are questions of presentation, acquisition and data representation. In all these aspects efficiency is often also considered to be a factor, and sometimes it is a prime motivator in itself.

3.1 Presentation: Monophonic vs Polyphonic

Retrieving musical information from monophonic sources is seen as a basic level for any system, and in many cases, there is an assumption that such pieces of music are purely melodic. This is in fact not the case for all monophonic music, such as in the case of self-accompanying melodies defined by Selfridge-Field and described in Section 4.7 on page 64.

In a similar manner, it cannot be said in polyphonic music that each score voice represents only one musical line, or even that in itself it can be considered a complete musical entity. Despite this, there is some considerable work on tracing monophonic patterns in polyphonic sources, such as the work of Lemström *et al.* [71, 73, 70, 72, 83].

The important information to draw from this is that any effective system must not assume that the notes contained in a single voice are necessarily coherent in their entirety. Equivalently, multiple voices cannot be considered to be separate, and frequently, coherence must be accepted as the interdependence of components of several voices.

3.2 Acquisition

In order to perform any kind of computation on a musical source, the information must be in a machine readable format. The various natures of these formats are discussed in Section 3.3, but here consideration is given to existing and emerging methods for transliterating from the three extant musical instances; performance, human instruction set, other instruction set. It is important to consider these as it will be clear that all three can introduce problems for an

analyst system if care is not taken.

3.2.1 Performance Translation

A musical performance, whether live or recorded, contains a richness of information including instrumentation, basic pitch, and gestural inflection. Each of these aspects is considered and their current status assessed. The trained human ear is capable of identifying the majority of musical sources when performed monophonically, though we would expect this to fall as the number of concurrent sources increases. No currently operational systems are capable of transcribing music performed simultaneously on multiple instruments. At the time of writing then, such systems are not sufficiently mature or useful to be considered.

An alternative way to capture information from a performance is to have the performers use specialist instruments which are capable of relaying aspects of the performance electronically. MIDI keyboards can relate note pitch exactly, as well as time and pressure data to a good degree of accuracy. MIDI controllers also exist in wind and brass forms, as well as transducers for stringed instruments. In general though, these "instruments" will play differently to a more authentic acoustic instrument, which inevitably impacts and alters the performance.

3.2.2 Score Translation

The translation of scores to a machine readable format without significant user interaction requires scanning of physical documents. The process of translating the score from graphical data is dependent to an extent on the desired use of the

score. If the system is to be able to reproduce the score, then relative graphical locations of score objects is relevant, but for some systems, simply acquiring timing and pitch data mat be sufficient.

Commercial software solution such as Neuratron's *Photoscore* [5] and Recordare's *SharpEye* [6] are capable of accurate results and have the ability to transfer data in formats including NIFF, Music XML and MIDI. There are also open-sourced applications, which would lend themselves more freely to an integrated system, such as Audiveris [1] and Gamera [77].

3.2.3 Machine Code Translation

There are two factors governing the interchange of machine format musical data. Provided that complete documentation of a format is in the public domain, the only limitation is the richness of data encapsulated in the source and destination formats. If the source contains only a subset of the information suitable for the destination, then either the missing data can be inferred, introducing potential errors, or it must be omitted entirely. Translation from MIDI to a format which is capable of chromatic differentiation (e.g. between $C\sharp$ and $D\flat$) will almost certainly introduce notational errors in all but the simplest music, for example.

While commercial data formats are not generally freely available, some have been inferred by analysing data files, such as the Finale *Enigma* format [28]. This may also be prone to error if not all features can be inferred.

3.3 Data representation

The musical score has evolved for over a millennia to be a succinct but complete graphical set of instructions to performers. Transferring the data into a form suitable for computation presents a number of non-trivial problems, which can, if not well handled, make meaningful analysis of the data impossible. The problem is in hierachicising elements when both horizontal (continuous time) and vertical (discrete time) groupings are possible.

3.3.1 Timing

Each sounding event has two temporal properties, an onset time and a duration. Durations can normally be expressed as fractions of some basic unit, and onset time is then the summation of all preceding sounding events. Bar lines are containers for notes with proscribed content length. As well as making navigation of a piece easier, they also introduce a framework for rhythmic stress.

3.3.2 Pitch

Ideally any system for music representation will be extensible to allow any pitch system, but even with CMN[3.1], there are a number of systems for allocating pitches. The pursuit of harmonious tunings is well documented[65]. Even if several notes share the same sounding pitch, the function of that sounding pitch will be inferred differently according to the individual notations. Thus, for CMN, it seems necessary to differentiate in data chromatic alteration of each letter named note.

[3.1] Common Music Notation, generally taken to cover the nomenclature used during the 18th-20th Centuries.

3.4 Efficiency

Generally speaking, if a task can be completed more quickly than was previously possible, this is a desirable outcome. The extent to which efficiency can be said to be a determinant of the usability of a system, and whether improving efficiency will noticeably improve the end-user experience of a system is a function of a variety of factors.

However, the specifics of the application are relevant. If a user wishes to download a ring-tone based on something they hummed, they will expect the answer quite quickly. If you want to analyse a symphony, that might have taken months by hand, then a day would be quite acceptable. So while efficiency is desirable, as long as the process is significantly quicker than existing methods, it is not essential.

Consideration should also be made in terms of balancing the amount of time improving the efficiency or speed of a system, both against whether that time might be better spent improving functionality, and whether the user-base will notice or appreciate any improvements in system performance.

The efficiency of a computational system can also be expressed in its ability to combine several aspects of analysis. A single monolithic system can acquire, analyse and annotate in a single pass.

3.5 Extending the use of Computation in Music Research

Although not directly impacting upon the work of this thesis, it seems that there are some areas in which methods and systems can progress to provide a more valuable resource to end users.

3.5.1 Recycling Queries

A review of the literature provides no evidence that processed queries are used to enhance system performance. Features which users have come to expect from commercial and semi-commercial systems information retrieval systems, such as Internet search-engines and on-line retail-outlets, are nowhere to be seen in MIR systems. That MIR systems have not, on the whole, achieved commercial deployment, should not be a barrier to the development of enhanced functionality.

3.5.1.1 Caching

Caching is basic data handling technique, but its optimisation can be complex. At the very least, a system should cache the queries made by a user for the duration of a session. Depending on the size of the archive, the number of users, and the frequency of requests, results could be cached by exact query.

3.5.1.2 Query Correction

The functionality of a search engine that says "did you mean .....?" may seem an impossibility as we have already established that in the right circumstances any sequence of notes can form a valid piece of music.. How should a system

respond if a note falls midway between two distinct pitches? Should a system search for *both* possible outcomes? For a small archive, this might only require a few seconds extra processing time, and for an efficient search algorithm should not double the search time. It may well return results that the user does not want, and it might be considered whether it will take much longer for that user to wade through all the unwanted results, than for the system to ask "Did you mean <plays on option> or <plays other option>?". Analysing this data could allow systems to make informed decisions about likely user errors.

3.6 Summary

A useful system cannot absolutely consider notes grouped by voice to be discrete streams of musical information. There will be cases where this is a valid assumption, but allowing for structural elements to be distributed between voices does not rule out that possibility, so this is the approach that must be adopted.

Using a computerised analog of a score appears to be the optimal solution for acquiring music for analysis. Not only does it contain the least cultural and interpretative context, but it provides a platform on which display the results of any analysis. It might be acceptable to use some other method to generate a score, such as machine listening, but in all probability this would require verification against a source score.

Chapter 4

Functional Programming and Parsing

The principal work presented in this thesis is that of applying parser techniques implemented in a functional language to problems of music analysis. Following the review of the history of traditional music analysis techniques in Chapter 2, this chapter will provide a description of both parsing and functional language development; both have a somewhat shorter history than that of music analysis.

4.1 Functional Programming

Functional programming is a methodology for programming computational devices whereby functions are used as components of a larger system. This contrasts with Imperative Languages (Fortran, COBOL, BASIC, Algol, Pascal, C, Ada), which describe the computational process in terms of program state. Object-Oriented imperative languages (Simula67, Smalltalk, C++, Java) are

characterised by program objects which are capable of receiving, processing, and sending data to other objects. Functional programming languages belong to the collective group of Declarative languages, alongside Logic Programming languages (Prolog, Mercury).

4.1.1 Motivations for Employing Functional Programming

An excellent summary of the advantageous features of functional programming can be found in Hughes [59]. While many features are not essential for specific musical tasks, they facilitate the process of programming the type of system envisaged in this thesis. Specifically, the abilities of functional languages to join functional micro-solutions using simple combinators allows programming problems to be subdivided more comprehensively and with fewer constraints. Additionally, the strong grounding of functional languages in mathematical notation makes them simple to learn and comprehensible for novice programmers. This is of course a concern if the work of this thesis is to become widely distributed, understood and expanded by its intended audience in the musical community.

4.1.2 Development of Functional Languages

A good description of the evolution of functional languages can be found in Hudak [55]. A brief chronology of this progression gives a picture of the discipline's development. Since the language used in this thesis, Haskell, is described in some detail in Section 4.1.3, but it is prescient to look at three of the major landmarks in it's development.

4.1.2.1 Lambda-Calculus

This was principally the work of Alonzo Church [31], though included contributions from his students Rosser and Kleene. The lambda calculus was the consistent subsystem that survived Church's ultimately unsuccessful aim of describing a formal system for the foundations of mathematics. In lambda calculus, every expression stands for a function with a single argument; the argument of the function is in turn a function with a single argument, and the value of the function is another function with a single argument. An overview of the continuing impact of Lambda-Calculus can be found in Barendregt [18].

As part of the system, a notational shorthand know as a *lamba abstraction* was introduced. Accordingly the notation $\lambda a.f$ denotes a function which takes an argument and returns the evaluation of the expression f with all free occurrences of a replaced by the actual argument. It is conceptually similar to function notations such as $f(x)$as seen in Figure 4.1.

Figure 4.1: Lambda and Function Notations

$$\begin{aligned} f(x) = x^3 + 2x^2 - 7x &\iff \lambda x.x^3 + 2x^2 - 7x \\ f(3) = 24 &\iff (\lambda x.x^3 + 2x^2 - 7x)3 = 24 \end{aligned}$$

All further uses of lambda notation in this thesis will use the notation more commonly used in current computing science literature, where the dot is replaced by a left-to-right arrow:$\lambda a \rightarrow f$.

4.1.2.2 LISP

McCarthy[80] borrowed Church's λ-notation to represent functions anonymously in his language LISP, which was the first functional[4.1] language to encapsulate the lambda abstraction:

> *"To use functions as arguments, one needs a notation for functions, and it seemed natural to use the λ-notation of Church. I didn't understand the rest of the book, so I wasn't tempted to try to implement his more general mechanism for defining functions."* [81]

Consider the LISP routine in Code 4.1 which calculates 2^4 by doubling 1 four times. Code is presented as at command line to a Common Lisp interpreter.

Code 4.1: LISP Doubling Function

```
[1] > (defun double (x)
(* 2 x))
[2] > (double (double (double (double 1))))
16
```

Explicitly typing the `double` function 4 times can be rewritten using a function `repeat` which apples the function `f` to the object `o` a total of `n` times as in Code 4.2.

Code 4.2: LISP Repeat Function

```
[3] > (defun repeat (f n o)
(if (zerop n)
   o
(repeat f (1- n) (funcall f o))))
[4] > (repeat #'double 4 1)
16
```

[4.1] LISP can be considered to be a multi-paradigmatic language, as it also encapsulates procedural and reflective aspects.

The syntax `#'` instructs LISP to interpret `double` as a function. Using the lambda expression in Code 4.3, the programme can avoid globally defining the function `double`.

Code 4.3: LISP Lambda Expression

$[5] > (repeat\ \#'(lambda\ (v)\ (*\ 2\ v))\ 4\ 1)$
16

McCarthy's paper also introduces a notation for conditional statements of the type *if... then...*, though written $(p_1 \rightarrow e_1, \cdots, p_n \rightarrow e_n)$, as well as the class of symbolic *S-expressions* used to order objects into lists.

4.1.2.3 FP

FP was presented as *"a class of simple applicative programming systems"* by Backus during his lecture to the ACM on his award of the Turing Award in 1977 [14]. As well as presenting a critique of von Neumann oriented languages and the traditional programming methods of passing a word-at-a-time through the *"von Neuman bottleneck"*, this paper delivers a prototype for a functional language with functions of a single type, a much reduced scope than that of a lambda-calculus system.

4.1.3 Haskell, a Popular Functional Language

Haskell is a functional programming language named after the US mathematician and logician Haskell Brooks Curry. The definitions for the language were laid out firstly in the Haskell Report [58], and later in the Haskell 98 Report [64]. The language is currently supported by a number of compilers (GHC [78] and NHC [96]), interpreters (Hugs [63]), and development environments (as extensions to the KDevelop and Eclipse IDE's). Some of its advantageous features

include recursion, list comprehension, pattern matching and guard statements, and it is currently one of the functional languages upon which, and with which, much research is being carried out.

Haskell was designed as a solution to a perceived problem that there were in fact too many disparate functional languages, with seemingly every major centre of research having their own language, as described in Hudak [55]. It was felt that academic research into functional languages was being hindered by the lack of a common language, even though the underpinnings of the distinct languages were similar.

For the reader unfamiliar either with functional programming as a paradigm, or specifically with the Haskell syntax, a good introduction can be found in Thompson [106]. A complete list of the software used in this thesis, as well as version details and availability at the time of publishing, can be found in Appendix B on page 172.

There has also been some considerable work implementing monadic parser combinators in Haskell, which is discussed further in Section 4.3 on page 56. Since this is a suitable method for constructing the parsers required by this thesis, it is expedient to use a language in which some considerable research into parser-combination has been conducted.

4.1.4 Functional Programming in Music Research

Of the attempts to integrate computation into music research, only a small minority have been implemented in functional languages. This is most probably due to functional languages being generally less well known than imperative

languages, particularly outside the field of computer science. It seems reasonable to suggest therefore that where functional languages are deployed in interdisciplinary research such as music, it is as a result of an existing expertise in the field on the part of the researchers. The research recounted in the sections 4.1.4.1 to 4.1.4.4 not only invokes functional programming in music research, but has also produced some of the more novel research in the field.

4.1.4.1 Smoliar

Smoliar [103] incorporated heavy Schenkerian influences into his (LISP) software. The software itself was intended for use by an operator with a deep understanding of music theory, and this is reflected in the language employed in the accompanying paper. It represents a step away from the *"reams of statistics"* and *"long strings of characters"* characteristic of preceding work. The paper discusses the the difficulty of establishing the function or significance, despite the tractability of extracting grammatical information. The internal format for notes contains only pitch information; a diatonic pitch symbol with optional prefixed chromaticisms and an integer register modifier. Notes can exist singularly, and there are two further grouping structures corresponding to simultaneous and sequential events which can be compounded indefinitely.

4.1.4.2 Dannenberg

Dannenberg's early work with functional languages was focused in the area of content generation. With Arctic [42], a Lisp-like language, higher-order functions, referred to as *prototypes*, can be described that respond to continuous (i.e. time based) input. These *prototypes* can become *instances* once various arguments such as start time, duration and envelope parameters are supplied.

Canon[41] was conceived as a hybrid between a music notation package and a programming language. In common with Artic it is based upon Lisp, and also adopts the methodology of higher-order functions, which can be used to alter anything up to the entire score. The notion of simultaneous and sequential note events, which may be repeatedly nested, is almost identical to that employed in Smoliar's system.

4.1.4.3 Cope

Cope's major contribution has been his work on Experiments in Musical Intelligence (EMI) [37, 38]. Written in LISP, it uses a rule based generator and a stylistic dictionary and has produced imitative works in the styles of Bach, Beethoven, Brahms, Bartok and others. Cope employs Automated Transition Networks in conjunction with 5 musical identifiers (Statement, Preparation, Extension, Antecedents, Consequent). Using these, music can either be rearranged (variation) or extended into larger structures.

4.1.4.4 Hudak

Hudak has been one of the proponents of the Haskell language, and his Haskore [57, 56] "Algebra of Music" is therefore very well defined in computational terms. Haskore allows musical objects to be generated and reused and therefore makes explicit some parts of the compositional process. The notation of tuples is also well handled by the use of fractions, thus avoiding irrational numbers, and a recursive notation is adopted for lines of music. A similar system to Hudak's has been produced by Akesson [11] but its existence is not widely known.

4.1.4.5 Comparison to the Work of this Thesis

Across these systems we can see glimpses of the power of the functional paradigm (Smoliar, in his nested data format; Cope, in confounding many lay listeners), but none of these systems has a significant overlap with the work presented here. Dannenberg and Hudak are not providing for analysis in their tools at all, being aimed squarely at generative (compositional) musical activities, though in that, they can highlight the utility of a nested data structure. Cope and EMI are performing some analysis, but its results are not directly exposed to the end user, rather they are used as a footing to create new music, and it is questionable what degree of influence the analysis has, since the system is already generating to a set of rules. The system that Smoliar produced is perhaps closest to the present work in spirit, though it is very highly specified to automating one particular form of reduction, and is not readily extensible, by design or accident, to other methodologies.

4.2 The Function of a Parser

A parser is a procedure that analyses the grammatical structure of its input with respect to some formal grammar. Human perception is a process of parsing, ordering and structuring the information received through our senses according to our knowledge.

A formal grammar is defined as a set of rules to define a set of strings of finite length using a (usually) finite alphabet. An analytic grammar describes a parser for a language. It reduces the input until it can decide whether or not a string belongs to the language. A generative grammar expands a starting token to create all the possible strings which make up the language.

The reader requiring a more thorough review of parsing techniques is referred to [62], though it should be appreciated that the absence of a tightly defined grammar for music reduces its direct impact.

4.3 Function and Uses of a Monad in Parsing

Many fields have claimed the word monad, though the meaning and derivation of the word can be quite different. Originally (coming from the Greek *monas*) the word meant an indivisible and primary unit, one from which all others are derived. Later it would become a synonym for the atom, and in the 19th Century, the subject of Leibniz's monadism. Leibniz's monads are simple forces which could not be affected by external physical forces but capable of internal activity. There are two understandings of monad within mathematics; In category theory a monad is a function object together with two associated natural transformations, while a division of predicate calculus in which predicate letters take only one argument is also termed monadic. There is also a musical use of monad, a single note, as distinct from the dyad (two notes) and triad (three notes).

In terms of functional programming, a Monad is a method for temporally ordering commands so that results can be passed along a sequence of execution. Wadler [109], building on the work of Moggi [85], was the first to introduce monad comprehension for programming languages. A minimal programmatic typing of a Monad can be found within the haskell98 libraries, as shown in Code 4.4 on the following page. The similarity between this specification and the parsers presented below and later in Chapter 5 should be obvious to the

reader.

Code 4.4: Haskell 98 Monad Definition

```
class Monad m where
(>>=) :: m a -> (a -> m b) -> m b
(>>) :: m a -> m b -> m b
return :: a -> m a
fail :: String -> m a
```

In typed haskell programs the λ character must be replaced with the more readily available \, though this thesis will use the λ character in its code samples to emphasise the nature of the construction.

Hutton and Meijer [61] describe their paper as *"a first introduction to the use of monads in programming"*, and it forms an excellent grounding in the standard use of Monadic Parser Combinators. A haskell implementation of a parser for text using combinators is provided here as an aide, beginning with Code 4.5.

Code 4.5: Simple Haskell Text Parser Primitives

```
type TP a =String -> [(a, String)]


pass   :: a -> TP a
pass a =λ inp -> [(a, inp)]


discard :: TP a
discard =λ inp -> [ ]


bind       :: TP a -> (a -> TP b) -> TP b
p `bind` f =λ inp -> concat [f v inp' | (v, inp') <- p inp]
```

TP is a function which takes a *String* of characters as its input and returns a

list of tuples.

- The function *pass a* succeeds[4.2] without consuming any of the *String* argument and returns the tuple consisting of a and the argument.
- The function *discard* fails[4.3], regardless of the argument, and returns an empty list.
- The *bind* function is monadic sequencing combinator. f is any function that takes a value and returns a *TP*. f can be applied to each of the tuples returned by p, which results in a list of lists of tuples. This is flattened into a single list of tuples by the inbuilt function *concat*.

These parsing primitives are of little use in and of themselves, but are necessary to produce the more useful functions in Code 4.6.

Code 4.6: Useful Text Parsers

```
item :: TP Char
item =λ inp → case inp of
                  [] → []
                  (x : xs) → [(x, xs)]


satsify  :: (Char → Bool) → TP Char
satisfy p =item `bind`
          λ x → if p x then pass x else discard


letter :: TP Char
letter =satisfy isAlpha


space :: TP Char
space =satisfy isSpace
```

[4.2] always returns a non-empty result
[4.3] always returns an empty result

The `letter` parser is a qualified `item` parser, with the caveat that the consumed character is a member of the alphabet. The operation of the `letter` parser can be seen in Figure 4.2.

Figure 4.2: Function of the 'letter' Parser

```
~$ letter "forty"

[('f',"orty")]

~$ letter "40"

[]
```

Further combinators are added in Code 4.7 on the following page which allow the parsing of larger structures. The operation of these combinated parsers is illustrated in Figure 4.3.

Figure 4.3: Function of Combinated Text Parsers

```
~$ word "forty thieves"

[("forty"," thieves")]

~$ some (word `skip` space) "Ali Baba and the forty thieves"

[(["Ali","Baba","and","the","forty","thieves"], "")]
```

Code 4.7: Extra Combinators

```
alt       :: TP a → TP a → TP a
p `alt` q =λ inp → ( p inp ++ q inp)


choice       :: TP a → TP a → TP a →
p `choice` q =λ inp → case (p `alt` q) inp of
                          → [ ]
                          (x : _) → [x]


perhaps   :: TParser a → TParser [a]
perhaps p =(some p) `choice` pass []


some   :: TParser a → TParser [a]
some p =p `bind`
          λ x → perhaps p `bind`
          λ xs → pass (x : xs)


skip       :: TParser a → TParser b → TParser a
p `skip` q =p `bind`
            λ x → perhaps q `bind`
            λ _ → pass x


word :: TParser String
word =some letter
```

4.4 Music and Formal Grammar

Taking music as a whole, it is initially difficult to reconcile it with the tenets of linguistic formalism. Pitch is continuously variable, though at least its range may be limited by the absolutes of human hearing[4.4]. Similarly, the temporal domain is continuously variable, even to the extent of being outwith the range of

[4.4]One could speculate as to whether sounds we cannot hear can form music; it is certainly true that the interaction of inaudible sounds can produce audible artefacts.

human perception[4.5]. Neither pitch nor duration then can be easily described as a *finite alphabet*, such as would be required for formalising a musical language, though attempts have been made to extend the representation beyond the simply linear[30].

If instead a solitary piece of music is considered, assuming it is of finite duration, there can only be a finite number of discrete temporal or pitch events. Even if an alphabet is defined by the product of these two sets, the product of our two finite sets must still be finite. Any single instance of music can be described in terms of a finite alphabet, but the power of any grammar formed over such an alphabet is constrained by the limited resources it defines.

It is possible to reduce the level of information required to describe a work of music. It would be simple to construct a 9 element alphabet upon a pitch basis of {*Lower, Same, Higher*} and a duration basis of {*Shorter, Same, Longer*}. Similar reduced descriptions have been shown to be effective in archival retrieval systems. In such instances, both query and target strings are available as continuous data streams, but the relationship between any one event and any other event not immediately juxtaposed cannot be established. This limited description is then only suitable for a limited subset of query types.

4.5 Disambiguation in Parser Design

While some languages have been designed to avoid ambiguity, many languages incorporate a degree of ambiguity. That is to say if the language is read naively, simply in terms of its syntactic elements, more than one semantic may be ex-

[4.5] One would undoubtedly die before hearing the entire 639 years of John Cages "As Slow As Possible"

tracted. In computational systems though, it is seen as being important to resolve these ambiguities, with good reason in some instances. Computer programmes must ultimately be unambiguous, so it may be necessary for a compiler or interpreter to disambiguate a number of potential parse trees into a single outcome. There are two common methods of disambiguation, described in Section 4.5.1 and Section 4.5.2.

4.5.1 Rule Precedence

Ordering the set of parsing rules creates a hierarchical precedence. This can be thought of either in the sense that the first successful rule is used, or that the rules that are successful are applied in a predetermined order. The BODMAS[4.6] rule for expanding mathematical terms should be well known to readers, and the precedence is as in Figure 4.4.

Figure 4.4: BODMAS precedence

1.Brackets
2.Orders
3.Division
4.Multiplication
5.Addition
6.Subtraction

The expression $7 + (6 \times 5^2 + 3)$ would then be resolved as in Figure 4.5.

Figure 4.5: BODMAS disambiguation

$$\begin{aligned} 7 + (6 \times 5^2 + 3) &= \\ &= 7 + (6 \times 25 + 3) \\ &= 7 + (150 + 3) \\ &= 7 + 153 \\ &= 160 \end{aligned}$$

[4.6] also BIDMAS and BEDMAS where I is Indices and E is Exponents

Aho [10] provides a more detailed and technical account for parsing ambiguous grammars without backtracking.

4.5.2 Application Length

Different parse rules will provide results of different lengths. Possible parse routes can then be ordered by length, typically with a preference for longer results. For example, in an English language parser

- $PR_{or}organic \Rightarrow or + ganic$ score 2
- $PR_{organ}organic \Rightarrow organ + ic$ score 5
- $PR_{organic}organic \Rightarrow organic$ score 7

though the absence of white space might already have been used to eliminate rules for *or* and *organ*.

4.6 Ambiguous Semantic

Multiple interpretations of music arise because it is often written to be deliberately ambiguous. In contrast, consider the following statement (attributed to Michael Crichton)

> *"The rules of grammar exist in large part to permit readers and writers to operate from a shared set of expectations"*

It seems reasonable that the natural languages we use every day should lend themselves toward a singularity of interpretation, since otherwise there should be great difficulty in inter-personal communication. This single factor should

impress upon the reader that any analysis system should not rely too heavily on linguistic theories, at least in as much as adopting them wholesale. It is understood that natural languages are ambiguous, but noted that they are frequently employed so as to produce an unambiguous result.

Consider the following: "I saw a man in the park with a telescope". Did the man whom I saw have a telescope, or did I use the telescope to see him. Was the man in the park when I saw him, was I, or indeed, were we both in the park. If we change the object; "I saw the moon in the park with a telescope", the sentence becomes somewhat less ambiguous, but only because the reader can eliminate some of the possible interpretations as being highly unlikely.

It can be shown though that at various points in its history, analysis has borrowed varying amounts of terminology and technique from linguistic theory. Terminology is more acceptable than technique, provided that we do not read too much into what extra properties may be imbued by referring to a piece of music as a *word* or *sentence*. If this terminology is taken too literally though, such as extrapolating that a musical sentence must retain properties of a grammatical sentence, techniques that have no place in music research can undermine any results.

4.7 Parsing Music

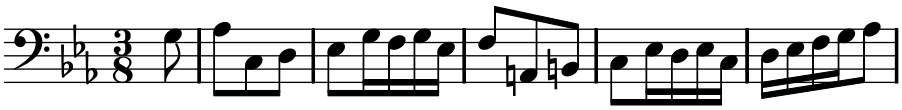
Selfridge-Field[101], shows 5 types of "disguised" melody. While these complications were originally recognised with respect to melody matching, they might equally be considered to disguise any musical feature or structure. Arguably, these all arise from the (incorrect) assumptions that voices represent discrete

streams of information, and that information will not only be limited to a single voice, but will be contiguous within that voice (see Section 3.1 on page 40). It is helpful though to see examples of the several ways in which a naive approach to parsing would be defeated.

4.7.1 Compound Melodies

Often in accompanied string writing, such as that of J. S. Bach in Figure 4.6, two or melodies can be compounded into a single instrumental line. The effect is one of dialogue, as if between more than one instrument. The problem is in ascertaining the themes of the dialogue and then extrapolating to which stream of the intercourse the notes or structures belong. This may be made more complex by the various voices speaking at different registers (which on a string instrument may require the use of a different string, with corresponding impact on the spectral tone), or the segments of a single voice following a particular pattern of movement.

Figure 4.6: Compound Melodies in Bach



4.7.2 Self-Accompanying Melodies

The tenet here is that some pitches perform functions as part of both the harmony and the melody, at the same time. This may be explicitly notated (such as in the Schubert excerpt cited by Selfridge-Field), using opposed stems where harmonic and melodic voices coincide. It may also be the case that the melody and harmony are compounded within the same part in an analog of 4.7.1. This

will almost certainly be the case where the scoring is for an instrument with limited polyphony. Where compounding pivots on a single note, this is rarely explicitly marked, though examples can be found throughout the Alexanian edition of the Bach 'cello suites.

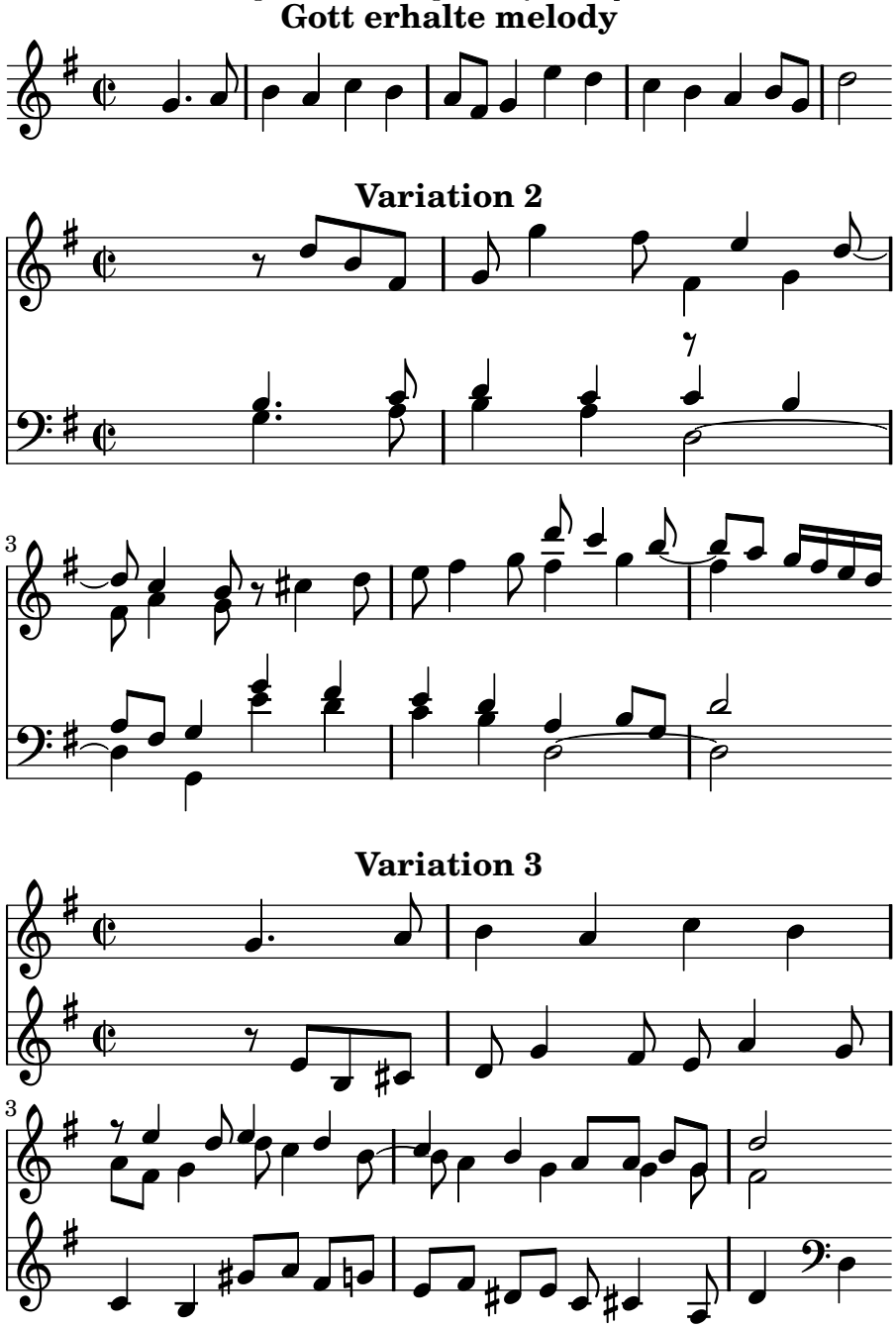
4.7.3 Submerged Melodies

Submerged melodies should not create problems for a thorough software implementation of an analytic system, since the disguise is perpetrated by positioning melodies within inner voices. This occlusion is really only effective to a listener, but can still defeat naive automated systems which assume that there is a vertical hierarchy from melody down to harmony.

4.7.4 Roving Melodies

The melody is distributed across more than one score part in this instance. This may mean that only incomplete instances of a query melody can be found in any single part, though it is also the case that parts may be "passing" a microstructure between them, in a "call and response" construction. Selfridge-Field uses an example from a Haydn keyboard Variation as shown in Figure 4.7.

Figure 4.7: Roving Melody Example



4.7.5 Distributed Melodiesduplet

Selfridge-Field suggests the distributed melody as one in which the notes of a melody are distributed one by one between two or more voices. This is much as with Section 4.7.4 but with melody fragmented into its atomic constituents. Such features are exceptionally difficult to hear as a listener, and are perhaps regarded as musical jokes, or exercises in compositional cryptography.

4.8 Summary

There are undoubtedly problems to be overcome in adapting traditional text and language parsing techniques to music. Some aspects have been addressed in a variety of extant research, but a complete solution requires all of this to be drawn together, and further extended, in order to overcome the issues outlined in this and the preceding two chapters.

Haskell should prove a good choice of language in which to build the analytic parser; it's active state of development means that support is available both for the developer and the end user who wishes to extend its functionality. The extensive literature on the use and implementation of monadic parser combinators also bodes well, as well as indicating (by it's absence) that this will be a novel extension of the technique. Exploiting the ability to create a set of results, i.e. not necessarily a single outcome, is also absent from the literature, which is readily explained by the typical usage patterns such as compilers which require a singular outcome.

Drawing together the examples from Section 4.7 on page 64, we can identify another feature which is not common to standard parsing systems that will be

necessary for a complete music-parsing system. The parsing solution must be able to identify interleaved streams of information within a single, and multiple voices, and this may very well require that some notes belong to several discovered structures. To allow atoms of the score (notes) to exist in multiple parsed elements will require a subtle but significant alteration to existing parsing techniques.

Chapter 5

Building a Basic Music Parser

The majority of software targeted at analysing music takes quite simple musical examples as its inputs. The extra difficulties involved in polyphonic analysis were discussed in Section 4.7 on page 64, and as a consequence the initial parsers in the system will be designed to look at simple monophonic musical input. This would normally be restricted to music where there is at most one sounding note at a given point in time, but polyphonic input is also included for the initial system, with the provisions that

1. There are no new notes that begin before any sounding notes have terminated
2. Notes that begin together also terminate together

Although simple, this first stage should allow the demonstration of at least a partial solution in extracting monophonic information patterns from partially polyphonic data sources. Functions not laid out explicitly in the body of the thesis can be found in Appendix A.1.

5.1 Basic Pitch Structures

An initial step in any system must be to define basic data types. Here, *Note* events are described as a duplet of integers, as shown in Code 5.1. Three types of `Note` are employed, which are described in the following sub- sections 5.1.1.1 to 5.1.1.3.

Code 5.1: Type Definition of a Note

$type\ Note = (Int, Int)$

5.1.1 Bases for Representing Notes

In the present system, each sounding event may be represented in any of the three bases; 7, 12 and 40. This allows the system to make use of the useful or unique properties of each basis. The three basis notations are homologous data types, which can be seen in Code 5.2.

Code 5.2: Type Definitions for the Three Note Bases

$type\ B40 = Note$
$type\ B12 = Note$
$type\ B7\ \ = Note$

5.1.1.1 Basis 40

Hewlett [51, 52] describes a system for describing notes uniquely using 40 locations within each octave. Only 35 locations are used, corresponding to 7 named notes modified by 5 accidental options, with the remaining 5 locations corresponding to null-events. In the allocation used in this system, the null-events are at locations 3, 9, 20, 26 & 32. Within a single octave, the table of allocations would be as in Table 5.1 on the next page.

Table 5.1: Base-40 Allocation

	A	B	C	D	E	F	G
♭♭	27	33	38	4	10	15	21
♭	28	34	39	5	11	16	22
♮	29	35	0	6	12	17	23
♯	30	36	1	7	13	18	24
x	31	37	2	8	14	19	25

Under this system, named intervals between notes always have the same integer value so that, for example, a perfect fifth is always formed between notes with an integer difference of 23. The system will not return valid results outside the circle of fifths described in the table, so that, for example, a perfect fifth above $B\mathbf{x}$ is undefined. Additionally, the register or octave within which a note is located must be recorded to allow for a thorough description of its pitch.

5.1.1.2 Basis 12

This system of note representation is analogous to twelve-tone or MIDI systems in having twelve unique events per octave. The limited number of locations per octave provides no way of differentiating between the notated pitches which may share the same basis-12 definition. The overlapping pitch names and their corresponding location are shown in Table 5.2 on the following page.

Table 5.2: Base-12 Allocation

Pitch	
$B\sharp/C\natural/D\flat\flat$	0
$B\mathbf{x}/C\sharp/D\flat$	1
$C\mathbf{x}/D\natural/E\flat\flat$	2
$D\sharp/E\flat/F\flat\flat$	3
$D\mathbf{x}/E\natural/F\flat$	4
$E\sharp/F\natural/G\flat\flat$	5
$E\mathbf{x}/F\sharp/G\flat$	6
$F\mathbf{x}/G\natural/A\flat\flat$	7
$G\sharp/A\flat$	8
$G\mathbf{x}/A\natural/B\flat\flat$	9
$A\sharp/B\flat/C\flat\flat$	10
$A\mathbf{x}/B\natural/C\flat$	11

5.1.1.3 Basis 7

Representing a note in a base-7 notation is the default behaviour of the Denemo score-editor. It is analogous to Curwen's[5.1] *Tonic Sol-Fa* system or pitch-name systems in that there seven unique names per octave. Denemo refers to this as a middle-c offset, and it may be useful to visualise this basis as corresponding to the number of lines and spaces a particular pitch is notated from middle-c. To describe a pitch fully, a representation of any accidental affecting the note is also required (-ve for each flat, 0 for natural or no accidental, and +ve for each sharps). No octave representation is required though both positive and negative integers are required to represent offsets above and below middle-c. As with basis-40 (Table 5.1 on the previous page), all notated pitches are represented uniquely, as can be seen in Table 5.3 on the following page.

[5.1] b Heckmondwike, Yorks., 14 Nov 1816; d Manchester, 26 May 1880

Table 5.3: Base-7 Allocation

	A	B	C	D	E	F	G
𝄫	-2,-2	-1,-2	0,-2	1,-2	2,-2	3,-2	4,-2
♭	-2,-1	-1,-1	0,-1	1,-1	2,-1	3,-1	4,-1
♮	-2,0	-1,0	0,0	1,0	2,0	3,0	4,0
♯	-2,1	-1,1	0,1	1,1	2,1	3,1	4,1
x	-2,2	-1,2	0,2	1,2	2,2	3,2	4,2

5.1.1.4 Summary of Bases

Table 5.4 shows the function of each of the two integers of a `Note` in each of the three bases, and how the pitch A♭, in the Great Octave, would appear under each representation.

Table 5.4: Summary of Note Bases

	base 7	base 12	base 40
A♭	(-9,-1)	(8,-2)	(28,-2)
integer 1	middle-c offset	chromatic step	basis 40 with c as 0
integer 2	accidental	octave	octave

As part of the system, functions are defined to convert from bases 7 and 40 to each of the other two bases, should this prove necessary. These functions use arrays to convert from integer, or integer pair, to string and back into an integer, or pair, corresponding to the new basis. A typical exchange function, from basis 40 to basis 7, might be employed as in Figure 5.1.

Figure 5.1: Basis Conversion Example

```
~$ forty2Seven (28,-2)

(-9,-1)
```

The functionality of these converters can be simply tested by applying them consecutively and checking that the input is the same as the output. There is

no error checking in these conversion functions at this point, it being presumed that the system will not pass invalid note descriptions, such as those arising from only 35 elements of the basis 40 system being assigned, to these routines. Figure 5.2 verifies the correspondence between two typical functions.

Figure 5.2: Verifying Basis Conversion

```
~$ seven2Forty . forty2Seven (28,-2)
(28,-2)
```

The point (.) combinator here implies the application of the function at the right of the point followed by the application of the left hand function. Neither basis 7 nor basis 40 notations can be recovered from basis 12 because of the lack of pitch or accidental identifiers, so there cannot be functions to perform such an operation (see Section 3.2.3 on page 42).

5.1.2 Note Groupings

Chords are formed from temporally co-incident *notes*, which can be from more than one score voice. Programmatically, a `Crd` is a list of `Note` elements, as shown in Code 5.3.

Code 5.3: Chord Type

$$type\ Crd\ =\ [Note]$$

It is important to observe here that the term *Chord* is used without many of its musical connotations. There is no suggestion that all the elements of a `Crd` must be coherent in any sense other than that already stated. One shortcoming of this system that will emerge is that the *Notes* of a *Chord* need not terminate co-incidentally. The problem of *Notes* that could temporally belong to more than

one vertical structure is non-trivial and an alternative solution is presented on page 115.

A *Sequence* is the only horizontal structure, consisting of two or more *Chords*. *Chords* are temporally ordered from left to right (first list element to last). A `Seq` is formed as a list of `Crd` elements, as shown in Code 5.5.

Code 5.4: Sequence Type

type Seq = [*Crd*]

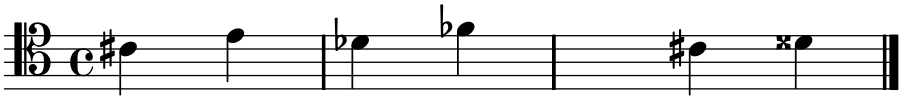
5.2 Pitch Operations

The ability to categorise the interval between two notes, and equally the ability to find the pitch produced by adding an interval to an initial pitch, should be an elementary and essential part of a music-analytical system. The solution provided also practically illustrates the greater depth of information provided by the base-40 and base-7 systems compared to the base-12 system.

If consideration is given to the interval between $C\sharp_4$and E_4, base-12 systems, such as MIDI, are limited to describing the interval as 3-semitones. In actual fact, such a system cannot differentiate $C\sharp_4$from $D\flat_4$, or E_4 from $F\flat_4$or $D\mathbf{x}_4$. In a system where those pitches are frequency-equivalent, this is less of a problem, though the *function* of an interval will still be different; $C\sharp_4$-$E_4$is a minor third, whilst $D\flat_4$-$E_4$is an augmented second . When performed on instruments capable of pitch inflection or adjustment, the frequency difference between the two intervals could amount to several cents. The notation of the pitch may also influence the players choice of execution; string players may open or close the hand in order to stretch forwards or backwards for a note. Figure 5.3 notates

three examples where the base-12 interval is identical, but the base-40 and base-7 descriptions are different in each case. In particular, we can see that only the third case describes stepwise or scale-like movement.

Figure 5.3: Basic Intervals



5.3 Parser Prototype and a Simple Parser

As the data types are now well defined, it is possible to define a parser based upon the specified input to the system. In contrast to most parsers in the literature, the input is not of the type `String` but of type `Seq`. The type definition for a generic music parser is given in Code 5.5.

Code 5.5: Type Definition of a Music Parser

$$type\ Parser\ a\ =\ Seq\ \rightarrow\ [(a, Seq)]$$

This means that a `Parser` of type `a`, takes a *sequence,* of type `Seq`, as an input and returns a set of tuplets, of type `(a,Seq)`, as its result.

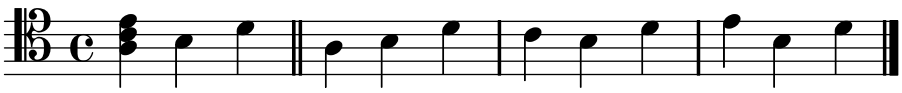
Replacing `B40` for the type `a`, an elemental parser for the atomic elements of a score, *Notes*, can be shown as in Code 5.6.

Code 5.6: An Item-like Note parser (cf. 4.6 on page 58)

$$\begin{aligned} &noteitem :: Parser\ B40 \\ &noteitem = \lambda\ inp\ \rightarrow\ \textbf{case}\ inp\ \textbf{of} \\ &\qquad\qquad [] \qquad\ \rightarrow [] \\ &\qquad\qquad (x : xs)\ \rightarrow [(xel, xs)\ \mid\ xel\ \leftarrow\ x] \end{aligned}$$

For each note `xel` of the leading chord `x`, this parser returns a `(xel,xs)` tuple, where `xs` is the sequence corresponding to the remainder of the input. In a typical parser for the elements of a textual input, there can be at most one member of the set of tuples returned, since there is only be one character at any point in a text stream input. Music, as mentioned in Section 4.4 on page 60, can have any number of temporally coincident notes. For this reason, the list returned by the `noteitem` parser will have as many pairs as there are notes in the chord at the head of the input. Thus in Figure 5.4, 3 results are produced corresponding to each note of the initial chord.

Figure 5.4: Parsing Polyphonic Input



```
~$ noteitem [[(29,-1),(0,0),(12,0)],[(35,-1)],[(6,0)]]

[((29,-1),[[(35,-1)],[(6,0)]]),((0,0),[[(35,-1)],[(6,0)]]),
((12,0),[[(35,-1)],[(6,0)]])]
```

If a text parser was to be implemented in this way, with the ability to handle multiple character choices at a single point of the text stream, it would be suitable for exploitation in systems where either the character is not sufficiently well defined, or a recognised glyph has more than one possible character attribution. It is suggested that this could potentially be of use in Optical Character Recognition systems, but a discussion is outwith the scope of this thesis.

5.4 Inter-Chord Relationships

Two consecutive chords, which may be formed from more than one score voice, can be related in one of two ways. In the first case, every note of the first chord is mapped onto a note in the second chord by a single relationship such as, for example, a rising major third. In the second case, it may be that only one note pairing between the chords satisfies, or is required to satisfy, a relationship. In Code 5.7, designed to recognise instances of the first case, `c1` and `c2` are chords and `f` would be the function that determines whether two notes conform to the required condition.

Code 5.7: Comprehensive Matching Function

$$
\begin{aligned}
&cMapAll \quad :: [a] \rightarrow [a] \rightarrow (a \rightarrow a \rightarrow Bool) \rightarrow Bool \\
&cMapAll\ c1\ c2\ f = and\ [\ or\ [\ f\ n1\ n2 \mid n2 \leftarrow c2] \mid n1 \leftarrow c1]
\end{aligned}
$$

This function will be `FALSE` unless for every note of `c1` there is a note of `c2` that provides a `TRUE` outcome for the function `f`. The second case can be useful for picking out a particular element from a rather dense structure, for example if the relationship were to be distributed between score voices. The routine to recognise instances of the second case can be formed as in Code 5.8.

Code 5.8: Partial Matching Function

$$
\begin{aligned}
&cMapAny \quad :: [a] \rightarrow [a] \rightarrow (a \rightarrow a \rightarrow Bool) \rightarrow Bool \\
&cMapAny\ c1\ c2\ f = or\ [\ or\ [\ f\ n1\ n2 \mid n2 \leftarrow c2] \mid n1 \leftarrow c1]
\end{aligned}
$$

This function will return a `TRUE` value if any note of the chord `c1` and any note of the chord `c2` provide a `TRUE` outcome for the function `f`. In both instances the function `f` would be of the form shown in Code 5.9 on the following page.

Code 5.9: Type of Function to be Matched

```
function ::B40 → B40 → Bool
```

5.5 Partially-Consumptive Parsers

As seen in Section 4.2 on page 55, parsers typically consume at least some part of their input. Hutton's `succeed` parser [60] is a parser that does not consume its input, but it does so unconditionally, corresponding to ϵ in BNF notation. A similar construct is found in most parsing systems. The `notescan` parser, presented in Code 5.10, requires no additional parameter. Additionally it specifically scans, but does not consume, a single note, the only condition being that there is sufficient input to scan. The provision for multiple notes in the leading chord, as for the `noteitem`, is maintained.

Code 5.10: Scanning Parser

```
notescan :: Parser B40
notescan =λ inp → case inp of
                    [] → []
                    (x : xs) → [(xel, x : xs) | xel ← x]
```

Clarification may be gained by examining the construction of a text parser with similar functionality, as shown in Code 5.11.

Code 5.11: Scanning Text Parser

```
type TxtParser a =String → [(a, String)]


itemscan :: TxtParser Char
itemscan =λ inp → case inp of
                    [] → []
                    (x : xs) → [(x, x : xs)]
```

Applying this parser on textual input would yield results such as those in Figure 5.5.

Figure 5.5: Scanning Text Parser in Operation

```
~$ itemscan "what is a parser"

[("w","what is a parser")]
```

The usefulness of such a parser for text applications will not be discussed but it seems likely that it could be applied to languages where words can share characters or glyphs.

5.5.1 Partial Consumption over Sets of Numbers

As this use of a non-consumptive parser is entirely novel, a further example is provided which illustrates a field of application. This example shows why this technique is necessary, and introduces some additional constraints when using elemental parsers in combination.

Consider a system that is required to deduce if a sequence of numbers is ascending and at what point in the input stream the numbers cease to ascend. The problem can be stated thus

> *Let there be some parser that takes some sequence of integers as an input and returns the sequence of integers which consecutively ascend by 1.*

$$\{u_i : u_i \in \mathbb{Z},\, u_{n+1} - u_n = 1\}$$

At the outset, the size of the set of integers which will be returned cannot be known, other than trivially that it can be no larger than the input set. Combinatorialy, the system can be required to parse each subsequent pair of integers for the condition $u_{n+1} - u_n = 1$, providing the last parse was successful. Assume that there is some traditionally consumptive parser `pairGT` that consumes a pair of integers if the difference between the first and second is exactly 1. The operation of such a parser is shown in Figure 5.6.

Figure 5.6: Parsing an Ascending Pair

```
~$ pairGT [1,2,3,4,5]
[((1,2),[3,4,5])]
```

The `pairGT` parser appears to function correctly, but needs to be repeatedly applied to parse the remainder of the input for the constraining condition. Let there also be some combinator `list`, whose operation is shown in Figure 5.7, that allows the system to repeatedly apply the `pairGT` parser until failure and flatten the results into a list.

Figure 5.7: Parsing Multiple Ascending Pairs

```
~$ list . pairGT [1,2,3,4,5]
[([1,2,3,4]),[5])
```

The problem which can be observed here is that while it is understood that the pair (4,5) satisfies the condition, it is not parsed as part of the condition satisfying sequence; only an even number of elements will be parsed. Looking at Figure 5.8 on the following page, a second problem can be observed.

Figure 5.8: The Problem of Discontinuity

```
~$ list . pairGT [8,9,6,7,4]
[([8,9,6,7],[4])]
```

There is no way of verifying the relationship between the $2n$ and $2n+1$ elements, which, in this instance, does not fulfil the condition. A non-consumptive parser is therefore used, as in Figure 5.9.

Figure 5.9: A Partially Consumptive Parser for Ascending Number Pairs

```
~$ pairGTpcp [1,2,3,4,5]
[((1,2),[2,3,4,5])]
```

If the same `list` combinator is used, the system will double-parse most of the elements, though it can now be seen, in Figure 5.10, that all elements may be consumed in a list with an odd number of elements satisfying the condition.

Figure 5.10: Side-Effect of Partial Consumption

```
~$ list . pairGTpcp [1,2,3,4,5]
[([1,2,2,3,3,4,4,5],[])]
```

Finally, a combinator must be introduced that disallows the duplication of the un-consumed elements. The operation of this combinator in conjunction with the partially consumptive parser provides exactly the outcome desired, as shown in Figure 5.11 on the following page.

Figure 5.11: Combining Partially Consumptive Parsers

```
~$ listpc . pairGTpcp [1,2,3,4,5,1]

[([1,2,3,4,5],[5,1])]
```

Now, the last element in the ascending sequence, though it does appear in the list of integers satisfying the condition, is not consumed. This may or may not be desirable. If the next operation were to look for sequences of numbers where $u_n - u_{n+1} = 1$ (i.e. descending), that element, (5) , may in fact belong to both condition satisfying sequences. A further parser to consume this hanging element could be added as a failure case for the `listpc` combinator if an element may only belong to one higher level structure. This would not generally be true in music since individual notes often belong to two or more overlapping structures.

5.6 Combining non-consumptive parsers to parse musical structures

For the purposes of parsing diatonic interval, accidentals may be ignored, so that a diatonic step is defined as notes which are one letter name apart. Sequential diatonic pitches therefore have a difference of 1 in basis-7. Whilst the underlying pitch notation is basis-40 , it is easy to convert this to a basis-7 notation corresponding to pitch name and accidental. At the highest level, the parser for a sequence satisfying some condition is formed as in Code 5.12 on the next page.

Code 5.12: Parser for a Sequence Satisfying some Boolean Condition

```
seqsat      :: Parser Seq → (Seq → Bool) → ABParser Seq
seqsat p c =p `bind` λ x → if c x then always x else never
```

Note that it is not necessary to explicitly state the length of the sequence which will be returned, neither is a constraint on individual elements of the sequence specified.

The combinators `somelap` and `manylap`, described below, combine partially consumptive parsers while the function `lap` ensures that the overlapping notes are not doubly represented in the output. The instance of `lap` in Code 5.13 allows for only one overlapping note at the head of each structure after the first.

Code 5.13: Combinators for Partially Consumptive Parsers

```
somelap    :: forall a. (Seq -> [([a], Seq)]) -> Seq -> [([a], Seq)]
somelap p =(p `pseq` manylap p) `using` lap


manylap    :: forall a. (Seq -> [([a], Seq)]) -> Seq -> [([a], Seq)]
manylap p =(somelap p) `choice` always []


lap          :: forall a. ([a], [a]) -> [a]
lap (xh, [])  =xh
lap (xh, xt) =xh ++ (tail xt)
```

5.6.1 Descending Parser

A partially consumptive parser for a pair of chords, `duplet`, can be employed with a predicate `seqstepdown`, whose result is `True` when at least one pair of notes within the chord pair is stepwise descending, to form `chordown`, as shown in Code 5.14 on the next page.

Code 5.14: Parser for Stepwise Descending Sequence

```
chordown  :: Parser Seq
chorddown =seqsat duplet seqstepdown
```

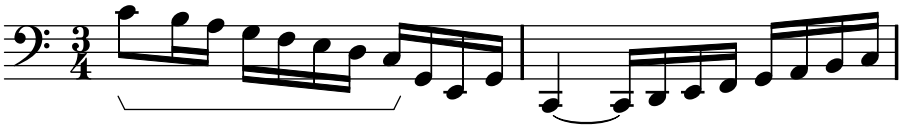
The length of the sequence returned is either two notes or zero, if the condition is not satisfied. In theory there is a shortcoming in this function. Errors will occur if the `b7interval` embedded in the `seqstepdown` function returns the value 1 but the basis-12 interval is either 0 (e.g. D♭ to C♯) or positive (e.g. F𝄫 to E♯). Practically, this problem has never arisen with the works input to the system, and the function could be re-written to check for this condition should this error be manifested during operation. A parser for simple stepwise descending structures can now be formed by repeatedly applying the `chordown` parser, as shown in Code 5.15.

Code 5.15: Parser for Descending Sequences

```
descent :: Parser Seq
descent =seqsat (somelap chorddown) length3
```

The additional constraint, `length3`, ensures that the result is not trivially short. Applying this parser to musical input (Bach unaccompanied 'cello suite) yields results as shown in figure 5.12, where the first 8 notes form a stepwise descending sequence.

Figure 5.12: Prélude BWV1009, Bars 1-2



```
$~ descent [[(0,0)],[(35,-1)],[(29,-1)],[(23,-1)],[(17,-1)],
[(12,-1)],[(6,-1)],[(0,-1)],[(23,-2)],[(12,-2)],[(23,-2)],
[(0,-2)],[(6,-2)], [(12,-2)],[(17,-2)],[(23,-2)],[(29,-2)],
[(35,-2)],[(0,-1)]]

[([[(0,0)],[(35,-1)],[(29,-1)],[(23,-1)],[(17,-1)],[(12,-1)],
[(6,-1)],[(0,-1)]],[[(0,-1)],[(23,-2)],[(12,-2)],[(23,-2)],
[(0,-2)],[(6,-2)],[(12,-2)],[(17,-2)],[(23,-2)],[(29,-2)],
[(35,-2)],[(0,-1)]])]
```

5.6.2 Ascending Parser

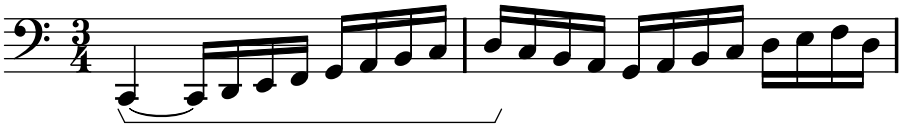
An equivalent parser for ascending stepwise sequences can be formed using the simple `chordup`parser and is shown in Code 5.16.

Code 5.16: Parser for ascending sequences

ascent :: *Parser Seq*
ascent =*seqsat* (*somelap chordup*) *length*3

When this parser is applied to the second and third bars of the Prelude from Bach's third 'cello suite, the results, in score and programmatic output forms, will be as in Figure 5.12.

Figure 5.13: Prélude BWV 1009, Bars 2-3



```
$~ascent [[(0,-2)],[(6,-2)],[(12,-2)],[(17,-2)],[(23,-2)],
[(29,-2)],[(35,-2)],[(0,-1)],[(6,-1)],[(0,-1)],[(35,-2)],
[(29,-2)],[(23,-2)],[(29,-2)],[(35,-2)],[(0,-1)],[(6,-1)],
[(12,-1)],[(17,-1)],[(6,-1)]]

[([[(0,-2)],[(6,-2)],[(12,-2)],[(17,-2)],[(23,-2)],[(29,-2)],
[(35,-2)],[(0,-1)],[(6,-1)]],[[(6,-1)],[(0,-1)],[(35,-2)],
[(29,-2)],[(23,-2)],[(29,-2)],[(35,-2)],[(0,-1)],[(6,-1)],
[(12,-1)],[(17,-1)],[(6,-1)]])]
```

5.6.3 Neighbour

The construction of a parser for a neighbour-note structure, is slightly more complex than the parsers described in Sections 5.6.1 and 5.6.2 since there are four forms that the structure can take. Looking at Code 5.17, `neighbourup` is a parser for an upper neighbour where the auxiliary note is one step above the object note.

Code 5.17: Parser Primitives for Neighbour-Note Structures

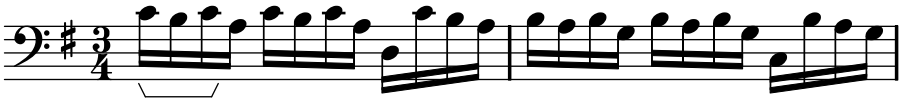
$$\begin{aligned} \mathit{neighbourup} &= ((\mathit{chordup}\ \text{`}\mathit{pseq}\text{`}\ \mathit{chorddown})\ \text{`}\mathit{using}\text{`}\ \mathit{lap})\ \text{`}\mathit{choice}\text{`}\ \mathit{chorddown} \\ \mathit{neighbourdown} &= ((\mathit{chorddown}\ \text{`}\mathit{pseq}\text{`}\ \mathit{chordup})\ \text{`}\mathit{using}\text{`}\ \mathit{lap})\ \text{`}\mathit{choice}\text{`}\ \mathit{chordup} \end{aligned}$$

The parsers for upper and lower neighbour structures can then be combined, as in Code 5.18 on the next page. In all cases the `choice` combinator acts by *choosing* the first successful parser, considering the left hand parser first, or failing if both parsers fail. The application of the parser to a musical example can be seen in Figure 5.14 on the following page.

Code 5.18: Parser for Neighbour-Note Structures

```
neighbour :: Parser Seq
neighbour =neighbourup `choice` neighbourdown
```

Figure 5.14: Courante BWV 1007, Bars 5-6



```
$~ neighbour [[(0,0)],[(35,-1)],[(0,0)],[(29,-1)],[(0,0)],
[(35,-1)],[(0,0)],[(29,-1)],[(6,-1)],[(0,0)],[(35,-1)],
[(29,-1)],[(35,-1)],[(29,-1)],[(35,-1)],[(23,-1)],[(35,-1)],
[(29,-1)],[(35,-1)],[(23,-1)],[(0,-1)],[(35,-1)],[(29,-1)],
[(23,-1)]]

[([[(0,0)],[(35,-1)],[(0,0)]],[[(0,0)],[(29,-1)],[(0,0)],
[(35,-1)],[(0,0)],[(29,-1)],[(6,-1)],[(0,0)],[(35,-1)],
[(29,-1)],[(35,-1)],[(29,-1)],[(35,-1)],[(23,-1)],[(35,-1)],
[(29,-1)],[(35,-1)],[(23,-1)],[(0,-1)],[(35,-1)],[(29,-1)],
[(23,-1)]])]
```

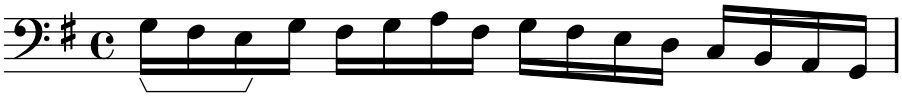
5.6.4 Passing

For completeness, the construction of a parser for the passing-note structure is included, though practically it can be replaced with a special instance of the parsers for ascending and descending structures. The primitives and final parser are shown in Code 5.19 on the next page and the application of the parser is shown in Figure 5.15 on the following page.

Code 5.19: Passing-Note Parser

```
passingdown =(chorddown `pseq` chorddown) `using` lap
passingup   =(chordup `pseq` chordup) `using` lap


passing :: Parser Seq
passing =passingup `choice` passingdown
```

Figure 5.15: Prélude BWV 1007, Bar 14



```
$~ passing [[(23,-1)],[(18,-1)],[(12,-1)],[(23,-1)],[(18,-1)],
[(23,-1)],[(29,-1)],[(18,-1)],[(23,-1)],[(18,-1)],[(12,-1)],
[(6,-1)],[(0,-1)],[(35,-2)],[(29,-2)],[(23,-2)]]

[([[(23,-1)],[(18,-1)],[(12,-1)]],[[(12,-1)],[(23,-1)],[(18,-1)],
[(23,-1)],[(29,-1)],[(18,-1)],[(23,-1)],[(18,-1)],[(12,-1)],
[(6,-1)],[(0,-1)],[(35,-2)],[(29,-2)],[(23,-2)]])]
```

5.7 Summary

Much of the utility and power of the system as laid out is mostly readily seen in the subsequent chapters, but we can already note some advancement. The most striking aspect is the ability to handle overlapping elements, as in the combination detailed in Code 5.17 on page 88. The ability to see disparate elements which hinge upon a single note is unique in the literature reviewed, as a result of the novel `lap` combinator. This allows for parsing the relationship between tokens, as well as the absolute value of tokens, which is vital for thorough and useful parsing of musical tokens.

The examples in this chapter show the system operating through a command line interface, a feature of interactive haskell development tools such as hugs [63] and ghci[78]. Developing software with a command-line interface can allow quick and easy debugging, but in order to provide a sterner test for the parsers, real musical examples should be available as inputs to the system. The utility of this interactive testing and development should not be lightly disregarded.

Having successfully proved the fundamentals of the system with single elemental parsers (and given the restrictions on page 70), the next chapters looks at enjoining these parts to parse entire musical passages.

Chapter 6

Integrating and Combining Parsers

In order to analyse an entire work, it will be necessary to increase the functionality of the parser combinators. As described in Chapter 5, the parsers currently within the system are each capable of detecting a single micro-structure at the head of the input. Clearly it is desirable to implement functionality to fulfil the following three applications.

1. The system can apply all the parsers and return the result(s) of those parsers that are non-empty (simultaneous application)
2. The system can then apply all the parsers again so that more than one set of results is returned (repeated application)
3. The system can bypass sections of the input where no micro-structures are found in order to apply the parsers to the whole of the input (application until completion)

Methods for these types of combination are described in Section 6.1.

Also in this chapter, the methodology used to interface the parsers with score files will be discussed, thus allowing results to be graphically represented. This is detailed in Sections 6.2 on page 99 and 6.3 on page 101 .

6.1 Parser Combination

Procedures to achieve combination of parsers are established in the literature (Section 4.2 on page 55), but some need to be reconsidered given the combination of partially-consumptive parsers and other factors.

6.1.1 Simultaneous Combination

It should be understood that a simultaneous application of parsers implies that they both operate over the same input. A typical text parser may use a `choice` or `alt` combinator to apply parsers simultaneously. The `choice` combinator will return at most one result, while the `alt` combinator, shown in Code 6.1 , may produce a more complex result.

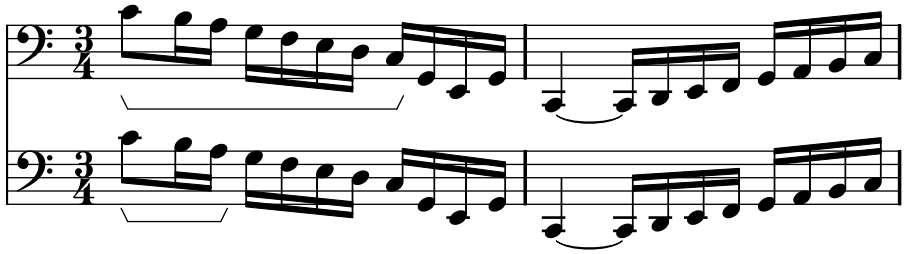
Code 6.1: Typical Form of the 'alt' Combinator

$$
\begin{aligned}
&alt \quad :: Parser\ a \rightarrow Parser\ a \rightarrow Parser\ a \\
&p\ `alt`\ q = \lambda inp \rightarrow (p\ inp +\!\!+ q\ inp)
\end{aligned}
$$

If both the parsers `p` and `q` are successful, the results are seen concatenated (the results are concatenated in every case, but this will not be obvious in the output if either `p inp` or `q inp` returns `[]`). Comparing Figure 6.1 with Figure 5.13, it can be seen that set of result now contains two `(Seq,Seq)` tuplets,

corresponding to the successes of the `descent` and `passing` parsers. The score markup is presented on two separate lines.

Figure 6.1: Prélude BWV1009, Bars 1-2



```
$~ (descent `alt` passing) [[(0,0)],[(35,-1)],[(29,-1)],[(23,-1)],
[(17,-1)], [(12,-1)],[(6,-1)],[(0,-1)],[(23,-2)],[(12,-2)],
[(23,-2)], [(0,-2)],[(6,-2)], [(12,-2)],[(17,-2)],[(23,-2)],
[(29,-2)], [(35,-2)],[(0,-1)]]

[([[(0,0)],[(35,-1)],[(29,-1)],[(23,-1)],[(17,-1)],[(12,-1)],
[(6,-1)],[(0,-1)]],[[(0,-1)],[(23,-2)],[(12,-2)],[(23,-2)],
[(0,-2)],[(6,-2)],[(12,-2)],[(17,-2)],[(23,-2)],[(29,-2)],
[(35,-2)],[(0,-1)]]),([[(0,0)],[(35,-1)],[(29,-1)]],[[(23,-1)],
[(17,-1)], [(12,-1)],[(6,-1)],[(0,-1)],[(23,-2)],[(12,-2)],
[(23,-2)],[(0,-2)],[(6,-2)], [(12,-2)],[(17,-2)],[(23,-2)],
[(29,-2)], [(35,-2)],[(0,-1)]])]
```

The `choice` parser, shown in Code 6.2 makes a 'decision' based on a precedence; a preference is established for one parse outcome over another for the leftmost successful parser. This is to say that that a system will 'choose' to try one parser after another and return the result corresponding to the first successful parser.

Code 6.2: Typical Form of the 'choice' Combinator

$$
\begin{aligned}
&\mathit{choice} && :: \mathit{Parser}\ a \rightarrow \mathit{Parser}\ a \rightarrow \mathit{Parser}\ a \\
&p\ `\mathit{choice}`\ q = \lambda\, \mathit{inp} \rightarrow && \mathbf{case}\ (p\ `\mathit{alt}`\ q)\ \mathit{inp}\ \mathbf{of} \\
&&& [] \rightarrow [] \\
&&& (x : xs) \rightarrow [x]
\end{aligned}
$$

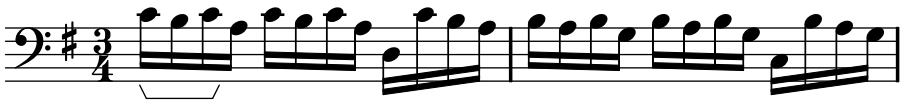
It is relatively simple to extend the `choice` parser to choose between any number of parsers, using an extended combinator such as `pllel` in Code 6.3.

Code 6.3: A Combinator for 'choice' over 2 or More Parsers

```
pllel    :: [Parser a] → Parser a
pllel ap = case (length ap) of
             2 → (head ap) `choice` (last ap)
             otherwise → (head ap) `choice` (pllel (tail ap))
```

Comparing Figure 5.14 on page 89 and Figure 6.2, it can be seen that the results are the same, which is the expected outcome since only the `neighbour` parser will succeed on this input.

Figure 6.2: Courante BWV 1007, Bars 5-6



```
$~ pllell [ascent,descent,neighbour] [[(0,0)],[(35,-1)],[(0,0)],
[(29,-1)],[(0,0)],[(35,-1)],[(0,0)],[(29,-1)],[(6,-1)],
[(0,0)],[(35,-1)],[(29,-1)],[(35,-1)],[(29,-1)],[(35,-1)],
[(23,-1)],[(35,-1)],[(29,-1)],[(35,-1)],[(23,-1)],[(0,-1)],
[(35,-1)],[(29,-1)],[(23,-1)]]

[([[(0,0)],[(35,-1)],[(0,0)]],[[(0,0)],[(29,-1)],[(0,0)],
[(35,-1)],[(0,0)],[(29,-1)],[(6,-1)],[(0,0)],[(35,-1)],
[(29,-1)],[(35,-1)],[(29,-1)],[(35,-1)],[(23,-1)],[(35,-1)],
[(29,-1)],[(35,-1)],[(23,-1)],[(0,-1)],[(35,-1)],[(29,-1)],
[(23,-1)]])]
```

6.1.2 Repeated Application

Combinators for repeated application of a parser were introduced in Code 5.11 on page 84, but this was in the context of a partially-consumptive parser for single structure. Here, consideration is given to repeated application of some

parser, which has been assembled through some form of combination, to recognise a number of structures. These combinators are shown in Code .

Code 6.4: Combinators for Repeating the Application of Parsers

```
many  :: Parser a → Parser [a]
many p =(some p) `choice` always []

some  :: Parser a → Parser [a]
some p =p `bind` λ x →
          many p `bind` λ xs →
          always (x : xs)
```

Neither the `ascent` or `descent` parser can usefully be repeated using these combinators. Since the final note of the result sequence is not consumed, and the closure of the results implies the next note pair not satisfying the condition, it is impossible for the parser to succeed again at that point.

The use of the `choice` parser may seem contrary to the stated purpose of avoiding predication rules (Section 4.5 on page 61). It is sufficient to note that the in the final system, such a predication will only be used where the rejected result is always a musical sub-result of the retained result. At this stage in the system, the nature of the parsers and the musical structures which they return ensures that disambiguation is rarely necessary.

Consider the example in Figure 6.3 on the following page which paraphrases a melodic theme used by Saint-Saëns in his *Le Carnaval des Animaux*. The parser repeatedly "chooses" which of the three parsers can succeed over the input. Conveniently all the elements overlap by exactly one note, so that the whole input is identified, even though the final note is not consumed.

Figure 6.3: Theme from Le Carnaval des Animaux

```
$~ many (pllel [descent, ascent, neighbour]) [[(0,0)],[(6,0)],
[(0,0)],[(6,0)],[(0,0)],[(6,0)],[(0,0)],[(6,0)],
[(0,0)],[(6,0)],[(12,0)],[(17,0)],[(23,0)],[(17,0)],
[(12,0],[(6,0)],[(0,0)],[(6,0)],[(12,0)],[(17,0)],
[(23,0)],[(17,0)],[(12,0)],[(6,0)]]

[([[[(0,0)],[(6,0)],[(0,0)]],[[(0,0)],[(6,0)],[(0,0)]],
[[(0,0)],[(6,0)],[(0,0)]],[[(0,0)],[(6,0)],[(0,0)]],
[[(0,0)],[(6,0)],[(12,0)],[(17,0)],[(23,0)]],[[(23,0)],
[(17,0)],[(12,0)],[(6,0)],[(0,0)]],[[(0,0)],[(6,0)],
[(12,0)],[(17,0)],[(23,0)]],[[(23,0)],[(17,0)],[(12,0)],
[(6,0)]]],[[(6,0)]])]
```

6.1.3 Complete input parsing

The set of parsers presented up to this point in the thesis is not claimed to parse every possible input to the system; the constituent parsers cannot be said to produce a generative grammar, nor are they intended to. It is therefore unlikely that the system so far presented, even under simultaneous and repeated application, will be able to parse a complete piece of music. Since it is useful to identify all occurrences of structure in a piece, this shortcoming must be overcome.

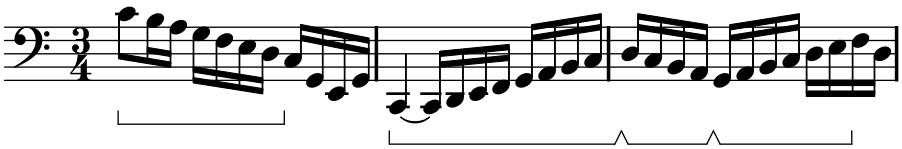
The `noteitem` parser has already been demonstrated as being able to consume a single note from its input. Raising `noteitem` to the level of a parser for sequences can be achieved as in Code 6.5 on the next page.

Code 6.5: Item-Like Parser for Sequences

$$
\begin{aligned}
&\mathit{seqitem} :: \mathit{Parser\ Seq}\\
&\mathit{seqitem} = \lambda\, \mathit{inp} \ \rightarrow\ \textbf{case}\ \mathit{inp}\ \textbf{of}\\
&\qquad\qquad\qquad\qquad [] \ \rightarrow\ []\\
&\qquad\qquad\qquad\qquad (x : xs) \ \rightarrow\ [([x], xs)]
\end{aligned}
$$

Adding the `seqitem` parser into the system as an option when all the other parsers have failed, it can be ensured that the whole input will eventually be consumed. Comparing Figure 6.4 to Figure 5.13 on page 88, and marking up only those results not produced by the `noteitem` parser, it can be seen that the system can continue to parse its input, even when no significant features are encountered.

Figure 6.4: Prélude BWV1009, Bars 1-3



```
$~ many(pllel[descent,ascent,neighbour,seqitem]) [[(0,0)],[(35,-1)],
[(29,-1)],[(23,-1)],[(17,-1)],[(12,-1)],[(6,-1)],[(0,-1)],
[(23,-2)],[(12,-2)],[(23,-2)],[(0,-2)],[(6,-2)],[(12,-2)],
[(17,-2)],[(23,-2)],[(29,-2)],[(35,-2)],[(0,-1)],[(6,-1)],
[(0,-1)],[(35,-2)],[(29,-2)],[(23,-2)],[(29,-2)],[(35,-2)]
,[(0,-1)],[(6,-1)],[(12,-1)],[(17,-1)],[(6,-1)]]

[([[[(0,0)],[(35,-1)],[(29,-1)],[(23,-1)],[(17,-1)],[(12,-1)],
[(6,-1)],[(0,-1)]],[[(0,-1)]],[[(23,-2)]],[[(12,-2)]],[[(23,-2)]],
[[(0,-2)],[(6,-2)],[(12,-2)],[(17,-2)],[(23,-2)],[(29,-2)],
[(35,-2)],[(0,-1)],[(6,-1)]],[[(6,-1)],[(0,-1)],[(35,-2)],
[(29,-2)],[(23,-2)]],[[(23,-2)],[(29,-2)],[(35,-2)],[(0,-1)],
[(6,-1)],[(12,-1)],[(17,-1)]],[[(17,-1)]],[[(6,-1)]]],[])]
```

6.2 Marking Up Results

As the number of individual parsers being applied to the input increases, the possibility of confusion as to the nature of specific sections of the results will arise. To surmount this problem, it is useful to assign a textual 'tag' to the result of each parser. This procedure is referred to in Hutton's description of lexical analysis [60]. Each lexical unit recovered by the parsing process is assigned a descriptive textual tag, and the pairing of the lexical unit and tag is termed a token. The whole process is encapsulated within a parser

6.2.1 Simple Parsers and Tags

In order to associate a tag with a parser, the two pieces of information are encapsulated within a *Token*. The type definition for a `Token` is provide in Code 6.6.

Code 6.6: Type Definition for a Token

$$type\ Token\ =\ (String,\ Seq)$$

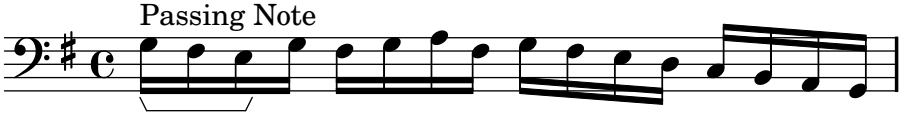
It is now possible to combine a parser for sequences with a textual tag to produce a parser for the `Token` type. The infix operator `tok` in Code 6.7 performs this combination.

Code 6.7: Attaching a Tag to the Results of a Parser

$$\begin{array}{l} tok \quad\quad :: Parser\ Seq\ \rightarrow\ String\ \rightarrow\ Parser\ Token \\ p\ `tok`\ t\ =\lambda\ inp\ \rightarrow\ [((t,x),xs)\mid(x,xs)\ \leftarrow\ p\ inp] \end{array}$$

When this parser is applied to an input, the result from the system will be as in Figure 6.5 on the next page, where it is now more clear as to the nature of the result.

Figure 6.5: Application of a Parser for Tokens




```
$~ (passing `tok` "Passing Note") [[(23,-1)],[(18,-1)],
[(12,-1)],[(23,-1)],[(18,-1)],[(23,-1)],[(29,-1)],[(18,-1)],
[(23,-1)],[(18,-1)],[(12,-1)],[(6,-1)],[(0,-1)],[(35,-2)],
[(29,-2)],[(23,-2)]]

[(("Passing Note",[[(23,-1)],[(18,-1)],[(12,-1)]]),[[(12,-1)],
[(23,-1)],[(18,-1)],[(23,-1)],[(29,-1)],[(18,-1)],[(23,-1)],
[(18,-1)],[(12,-1)],[(6,-1)],[(0,-1)],[(35,-2)],[(29,-2)],
[(23,-2)]])]
```

6.2.2 Tagging a Complete Parse

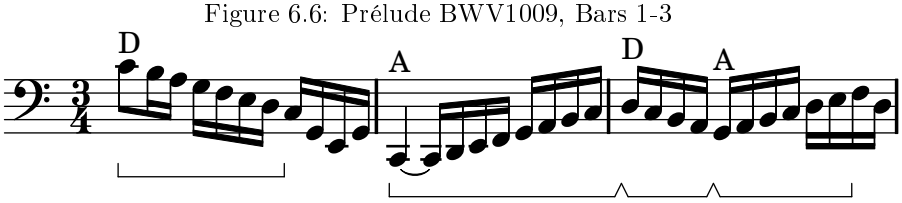
The tagging of a structure is the final process in parsing an input, so associating a group of parsers with the appropriate tag requires a new combinator, shown in Code 6.8.

Code 6.8: A Combinator for Parsers and Tags

$$
\begin{aligned}
&mlex :: [(Parser\ Seq,\ String)] \rightarrow Parser\ [Token] \\
&mlex = many\ .\ (foldr\ op\ never) \\
&\qquad where\ (p, t)\ `op`\ xs\ =\ (p\ `tok`\ t)\ `alt`\ xs
\end{aligned}
$$

In the final tagging example in Figure 6.6 on the next page, the results are similar to those of Figure 6.4, with the addition of the single character tags. The use of single character tags as compared to a more complete description is discussed in Section 6.3.

Figure 6.6: Prélude BWV1009, Bars 1-3




```
$~ mlex [(ascent, "A"),(descent,"D"),(neighbour, "N"),
(seqitem, "")][[(0,0)],[(35,-1)],[(29,-1)],[(23,-1)],[(17,-1)],
[(12,-1)],[(6,-1)],[(0,-1)],[(23,-2)],[(12,-2)],[(23,-2)],
[(0,-2)],[(6,-2)],[(12,-2)], [(17,-2)],[(23,-2)],[(29,-2)],
[(35,-2)],[(0,-1)],[(6,-1)], [(0,-1)],[(35,-2)],[(29,-2)],
[(23,-2)],[(29,-2)],[(35,-2)],[(0,-1)],[(6,-1)],[(12,-1)],
[(17,-1)],[(6,-1)]]

[([("D",[[(0,0)],[(35,-1)],[(29,-1)],[(23,-1)],[(17,-1)],
[(12,-1)],[(6,-1)],[(0,-1)]]),("",[[(0,-1)]]),("",[[(23,-2)]]),
("",[[(12,-2)]]),("",[[(23,-2)]]),("A",[[(0,-2)],[(6,-2)],
[(12,-2)],[(17,-2)],[(23,-2)],[(29,-2)],[(35,-2)],[(0,-1)],
[(6,-1)]]),("D",[[(6,-1)],[(0,-1)],[(35,-2)],[(29,-2)],
[(23,-2)]]),("A",[[(23,-2)],[(29,-2)],[(35,-2)],[(0,-1)],
[(6,-1)],[(12,-1)],[(17,-1)]]),("",[[(17,-1)]]),
("",[[(6,-1)]])],[])]
```

6.3 Displaying System Results

The thesis has to this point described methods for producing output at the command line. The processes involved in transforming these results into the graphical examples that have accompanied the preceding figures is now described.

6.3.1 Denemo

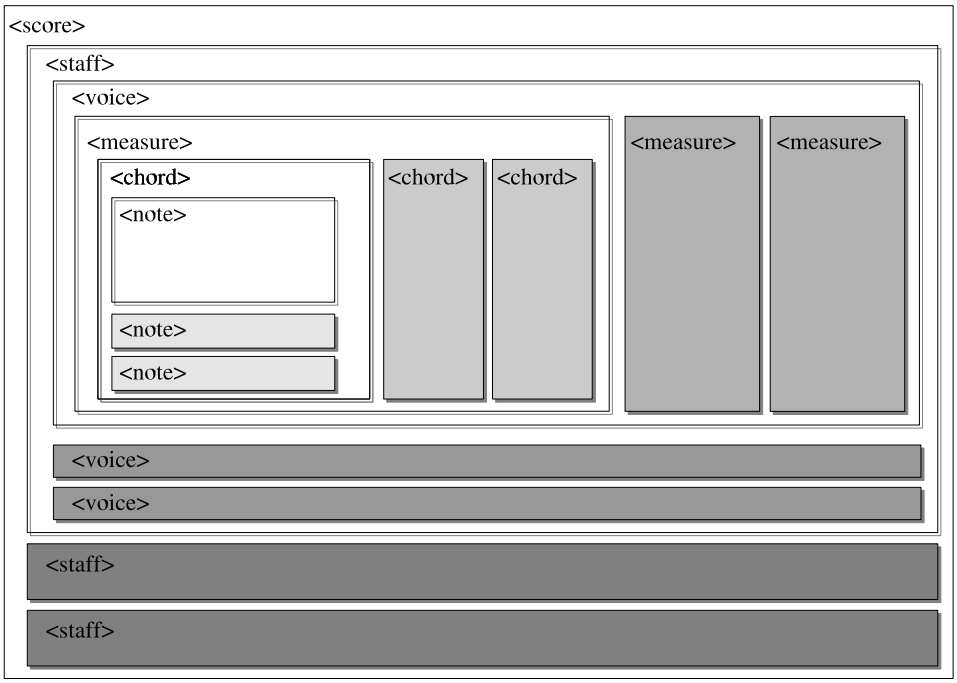
Denemo is a graphical front-end to the Lilypond score creation tool. It uses an XML schema, compressed with a Lempel-Ziv [110] algorithm, for storing score

data. It is therefore most facile to use an XML parser to transform the score files into the form readable by the analytic system. The HaXml toolkit was used for this purpose, though this required more integration than was initially envisaged. HaXml includes a tool for generating parsers from a valid XML DTD[6.1], but due to errors and inconsistencies in the Denemo DTD, and slight differences between the intended uses of HaXml and those employed in the system, this was not a viable option. The system for extracting information from the score file was therefore manually assembled using HaXml primitives.

The XML structure is such that the score is made up of staffs which contain voices. Each voice is constructed of measures, and each measure contains chords made up of notes. In order to merge a score into a single `Seq` type sequence, each voice is deconstructed into series of tuples, each containing a chord and an onset time. By comparing the onset times of the chords in each voice, the chords can be merged and sequenced into a coherent stream from which the onset times are removed to leave only the information required by the analytic parser. In the diagrammatic representation of the denemo data structure in Figure 6.7 on the following page, the shaded elements indicate that there may be one or more of that element within the parent element e.g. one or more notes in a chord. This is not a complete description but makes clear the hierarchical structure.

[6.1] Document Type Definition

Figure 6.7: Denemo Data Structure



Once tags have been assigned by the analytic system, it becomes necessary to mark this information and the structure groupings into the score file. To do this, it was necessary to uniquely identify each note so that all structures could be accurately described. Denemo includes within its format an *xml-id* for each event, so the unique identification of notes could simply be achieved by adding this information to the analytic data type for notes, as in Code 6.9.

Code 6.9: Note Data Type with Unique ID

$$type\ Note = (Int, Int, Int)$$

The part of the system which re-integrates the structural information into the Denemo file is not optimised for speed. It simply scans the entire score until the note with the correct *xml-id* is found. Clearly this can be improved but its

performance is not a barrier to the functionality of the system as a whole.

6.3.2 Lilypond

Simple markup of analytical features is not supported natively Denemo, which is to say that there is no internal data type exclusively corresponding to such a feature. Denemo does though provide for any Lilypond function to be included verbatim as a *lily-directive*. These features do not have a graphical representation within Denemo, but are rendered by Lilypond when that is used as the output engine.

Three Lilypond features have been assessed for use in marking out the structures within the score. None seems to be entirely suitable, but is within the spirit of the development of score notation that new ideas should eventually spawn new graphical forms, and it is to be hoped that systems such as this will impact upon the development of score-editing software.

6.3.2.1 Analysis Brackets

Analysis brackets were added to Lilypond at version 2.0, and so were not available at the commencement of the research for this thesis. Their advent seems at first an ideal solution, as they provide square edged brackets starting and ending on specified notes. Two problems became apparent with the routines for this feature. Firstly, a new bracket cannot begin on the same note as another ends because this forms a rogue one not bracket on the overlapping note. Secondly that although nesting is possible, it is not possible to have two layers of annotation unless one is a subset of the other. Text annotations to the bracket must be added separately, but this is easily achieved. Figures 6.1 on page 94, 6.2 on

page 95, and 6.5 use analysis brackets. In order to achieve full use for analysis brackets, each bracket should probably be uniquely specified so that start and end points cannot be confused.

6.3.2.2 Phrase Marks

Phrase marks are often used by composers to indicate intended performance structures. This means that they can only be used on un-articulated scores, which is somewhat limiting, and even then might be mistaken by the casual observer as slurs or some other original feature intended by the composer. Even on an unarticulated score textual accompaniment must still be added separately. There is some scope for representing more than one layer of analysis since Lilypond provides for a number of slurring and phrasing styles.

6.3.2.3 Pedal Brackets

Pedal Brackets are a feature intended for keyboard music. As with phrases and slurs, elements may end on the same note as a new bracket begins. Figures 6.3 on page 97, 6.4 on page 98, and 6.6 on page 101 use pedal brackets. Since a pedal is either down or up, and cannot therefore perform both functions at once, there is no need for the notation to support overlapping elements.

6.4 Summary

While simultaneous and repeated (points 1 & 2 on page 92)application of micro-parsing elements is a feature common to all parsing systems, the set-wise output of the monadic parser combinators described here allow us to take forward the result of all successful micro-parsers. This is vital to fulfilling the desire that we

avoid predication rules (page 1).

The 3rd point (application until completion) might be regarded as an expedient to achieving a pseudo-complete system. This status is somewhat altered by 7.1 on the next page, and is essential for the future expansions of the system; this will be a system that is not afraid to reveal its own shortcomings.

Chapter 7

Extension and Modification of the System

In this chapter, two additional parsers for structure are introduced, one of which infers structure and another which is more traditional in operation. Further modifications are made to the data structure to include more temporal information, and investigations are made into the temporal groupings of notes.

7.1 An Inferential Parser

In Chapter 5, 4 parsers for micro-structures were introduced, while the special case of a single note event to allow complete parsing of the input was introduced in Section 6.1.3 on page 97. When parsing some works, this is still sufficient to mark-up only a small percentage of the input, but for other pieces, they are mostly unsuccessful. One of the aims of this work is to limit the amount of *a priori* knowledge required to perform an in depth analysis, so it may be useful

for the system to provide some feedback as to what kinds of new structure might be useful in marking up a work.

Typically in a piece where only a small percentage of the content is constructed from the identified structures, large portions of the input will be parsed by the `seqitem` parser. Where only one `seqitem` separates each large structure, this implies only that the structures do not overlap, that the pitch distance between he ending of one structure and the beginning of the next is greater than one diatonic step. It would be possible to add a parser for a leap greater than one diatonic step, but this would be quite a general result and might still produce long sequences of only that result. It seems then that the system would benefit from the addition of a parser capable of consuming large portions of the input not consumed by other parsers.

7.1.1 Parsing for Failure

To resolve the problem presented in Section 7.1 on the previous page, a parser is created which succeeds where other specified parsers fail. The formation of this parser is shown in Code 7.1.

Code 7.1: Parser which succeeds on failure

$$\begin{aligned} &\mathit{interesting} \quad :: [\mathit{Parser\ Seq}] \rightarrow \mathit{Parser\ Seq} \\ &\mathit{interesting}\ p = \mathit{seqsat}\ (\mathit{pnone}\ p)\ \mathit{length} 3 \end{aligned}$$

`p` is the set of parsers which must all fail for the `interesting` parser to succeed. Additionally the length of the parsed input must be at least 3 notes long, since interesting structures with a length of only 2 notes will be diatonic leaps. The `pnone` parser in Code 7.2 on the next page is separated into two parts, with `noneagain` consuming a single `seqitem` before checking the success of the set of

parsers `p` using the `pnone` parser again, recursing one of the parsers succeeds.

Code 7.2: Parser primitives for success from failure

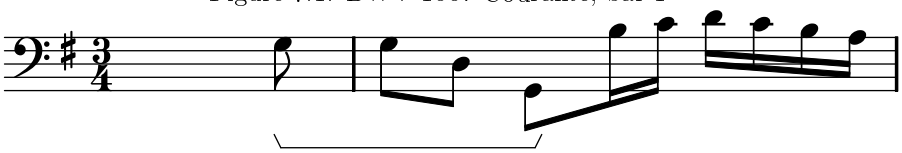
```
pnone    :: [Parser Seq] → Parser Seq
pnone p =λ inp → case ((pllel p) inp) of
                   [] → noneagain p inp
                   otherwise → always [] inp

noneagain    :: [Parser Seq] -> Parser Seq
noneagain p =seqitem `bind` λ y →
               (pnone p) `bind` λ z →
               always (y ++ z)
```

7.1.2 Using the Inferential Parser

The effect of the `interesting` parser is to return the longest continuous sequence of notes upon which all of a set of parsers fail. The rationale for calling this the `interesting` rather than the `inferential` parser is that the results often form interesting new structures.

In Figure 7.1 on the following page, the `interesting` parser can be seen in operation, succeeding over the four notes that begin the Courante from the first 'cello suite. In this instance, the interest is in outlining a tonic chord, and a similar motif is used for each of the first four bars to sketch a I-I-IV-V motion in G major (though this is not deduced by the system).

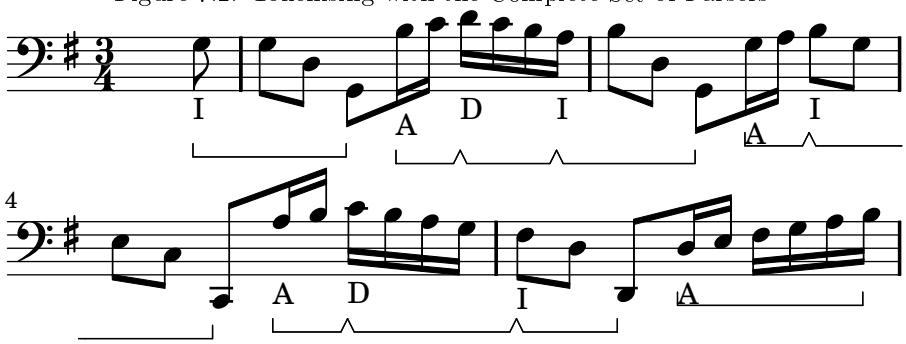
Figure 7.1: BWV 1007 Courante, bar 1



```
 ~$ (interesting [ascent,descent,neighbour]) [[(23,-1,6)],
[(23,-1,9)],[(6,-1,11)],[(23,-2,14)],[(35,-1,16)],[(0,0,18)],
[(6,0,21)],[(0,0,23)],[(35,-1,25)],[(29,-1,27)]]

[([[(23,-1,6)],[(23,-1,9)],[(6,-1,11)],[(23,-2,14)]],
[[(35,-1,16)],[(0,0,18)],[(6,0,21)],[(0,0,23)],[(35,-1,25)],
[(29,-1,27)]])]
```

The operation of the interesting parser differs slightly from the preceding structural parsers because it does not use partial consumption. The first note of an interesting result may belong to the result of a previously successful parser, but the last note identified by the `interesting` parser belongs only to the interesting result. This follows logically from the construction of the parser. Figure 7.2 on the next page is a complete parsing of the first 4 bars of the Courante used in Figure 7.1.

Figure 7.2: Tokenising with the Complete Set of Parsers




```
~$ mlex [((interesting [ascent,descent,neighbour]),"I"),
(ascent,"A"),(descent,"D"),(neighbour,"N")]
[[(23,-1,6)],[(23,-1,9)],[(6,-1,11)],[(23,-2,14)],[(35,-1,16)],
[(0,0,18)],[(6,0,21)],[(0,0,23)],[(35,-1,25)],[(29,-1,27)],
[(29,-1,27)],[(35,-1,30)],[(6,-1,32)],[(23,-2,35)],[(23,-1,37)],
[(29,-1,39)],[(35,-1,42)],[(23,-1,44)],[(12,-1,47)],[(0,-1,49)],
[(0,-2,51)],[(29,-1,54)],[(35,-1,56)],[(0,0,59)],[(35,-1,61)],
[(29,-1,63)],[(23,-1,65)],[(18,-1,68)],[(6,-1,70)],[(6,-2,73)],
[(6,-1,75)],[(12,-1,77)],[(18,-1,80)],[(23,-1,82)],[(29,-1,84)],
[(35,-1,86)]]

[([("I",[[(23,-1,6)],[(23,-1,9)],[(6,-1,11)],[(23,-2,14)]]),
("A",[[(35,-1,16)],[(0,0,18)],[(6,0,21)]]),("D",[[(6,0,21)],
[(0,0,23)],[(35,-1,25)],[(29,-1,27)]]),("I",[[(29,-1,27)],
[(35,-1,30)],[(6,-1,32)],[(23,-2,35)]]),("A",[[(23,-1,37)],
[(29,-1,39)],[(35,-1,42)]]),("I",[[(35,-1,42)],[(23,-1,44)],
[(12,-1,47)],[(0,-1,49)],[(0,-2,51)]]),("A",[[(29,-1,54)],
[(35,-1,56)],[(0,0,59)]]),("D",[[(0,0,59)],[(35,-1,61)],
[(29,-1,63)],[(23,-1,65)],[(18,-1,68)]]),("I",[[(18,-1,68)],
[(6,-1,70)],[(6,-2,73)]]),("A",[[(6,-1,75)],[(12,-1,77)],
[(18,-1,80)],[(23,-1,82)],[(29,-1,84)],[(35,-1,86)]])],
[[(35,-1,86)]])]
```

7.2 Alternation Parser

A common structure in the Bach cello suites is formed when a moving part is note-wise alternated with a static pitch. If the static pitch and moving part

were to be considered as different voices, as they are indeed notated in several editions of the suites, then the static pitch can be considered a pedal note. The parser for this structure is therefore termed `pedal`, though this may not be entirely musically accurate in every situation in which it is successful.

7.2.1 Construction of the Alternation Parser

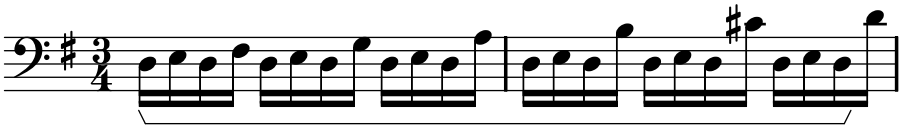
An alternation structure must be at least 5 notes in length to allow the formation of the pedal like action of the static pitch, for which 3 instances is the minimum. The `pedal` parser is formed from multiple instances of the `pedalpoint` parser, which is itself formed by checking a correctness function against three consecutive notes. The remaining functionality for this parser is described in Appendix A.2, while the top-level functions are shown in Code 7.3.

Code 7.3: Parser for Alternation

```
pedalpoint :: Parser Seq
pedalpoint =seqsat tripletnote pedalBin


pedal :: Parser Seq
pedal =seqsat (somelap pedalpoint) length5
```

The operation of the alternation parser is identified using the tag `Pd` so that no confusion is made with passing structures that were previously identified by the tag `P`. The function of the alternation parser is illustrated in Figure 7.3 on the following page. It can be noted that the final note of the input is not recognised as part of the alternation structure. It seems logical here that it should, but since this will not always be the case, it is reasonable for the parser to function as it does.

Figure 7.3: Alternation Example



```
 ~$ (pedal `tok` "Pd") [[(6,-1,300)],[(12,-1,302)],[(6,-1,304)],
[(18,-1,306)],[(6,-1,309)],[(12,-1,311)],[(6,-1,313)],
[(23,-1,315)],[(6,-1,318)],[(12,-1,320)],[(6,-1,322)],
[(29,-1,324)],[(6,-1,327)],[(12,-1,329)],[(6,-1,331)],
[(35,-1,333)],[(6,-1,336)],[(12,-1,338)],[(6,-1,340)],
[(1,0,342)],[(6,-1,345)],[(12,-1,347)],[(6,-1,349)]
,[(6,0,351)]]

[(("Pd",[[(6,-1,300)],[(12,-1,302)],[(6,-1,304)],[(18,-1,306)],
[(6,-1,309)],[(12,-1,311)],[(6,-1,313)],[(23,-1,315)],
[(6,-1,318)],[(12,-1,320)],[(6,-1,322)],[(29,-1,324)],
[(6,-1,327)],[(12,-1,329)],[(6,-1,331)],[(35,-1,333)],
[(6,-1,336)],[(12,-1,338)],[(6,-1,340)],[(1,0,342)],
[(6,-1,345)],[(12,-1,347)],[(6,-1,349)]]),[[(6,-1,349)],
[(6,0,351)]])]
```

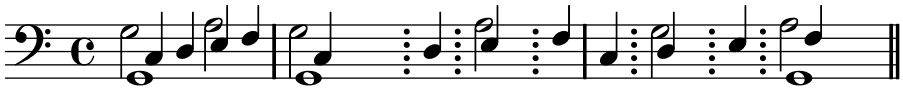
7.3 Temporally Ordering Polyphonic Input

All the examples of musical input to the system to date have been, if not monophonic, at least scored for a single instrument. There have therefore been few instances where multiple notes are sounding simultaneously. The method for grouping notes together into the structure has been based on onset times i.e. notes that begin together are grouped together. There are however two alternative methods for grouping notes.

7.3.1 Group by Termination

Notes can be grouped together according to their termination time. This is conceptually similar to the method outlined in the previous paragraph except that notes that end together are grouped together. This method will typically produce quite different vertical grouping structures, as shown in the example in Figure 7.4.

Figure 7.4: Temporal Grouping Strategies

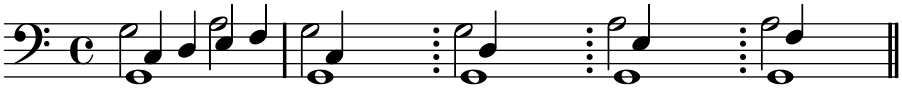


The first bar contains three voices; one using only a semi-breve, one using minims and the other using crotchets. The second bar shows the notes grouped by onset time, and the third bar show grouping by termination time. In both grouping instances, each note appears in only one vertical structure.

7.3.2 Group by Sounding

Notes can also be grouped into those that are sounding simultaneously. For this strategy to be completely implemented, notes must be allowed to exist in more than one vertical structure. This is the most complete solution to the problem of temporal organisation, but provides for some unwanted results. It would be possible for sustained notes to occur more than once in discovered structures, which is potentially undesirable. Recalling Figure 7.4, Figure 7.5 on the following page shows, in the second bar, how a grouping based on sounding tones would function.

Figure 7.5: Organisation by Sounding Tones



In the above examples, the number of groupings produced by each strategy was the same, but this need not be the case.

7.4 Modification of the Data Structure

Given the considerations presented in Section 7.3 on page 113, it can be considered beneficial for the system to contain temporal information for each note. This potentially allows the temporal structure to be changed mid-analysis, rather as the note-basis can be changed. Temporal information is stored in a fractional manner analogous to the American model, where a *crotchet* is termed a *quarter-note*. The data-type used for fractions is from Jan Skibinski's `Fraction` module [102]. Using fractions allows the exact representation of onset and duration times for tuplets, and is the method employed in Haskore [57].

7.4.1 Redefining the Data Structures

Each note will now be represented by a 5-tuplet, as in Code 7.4 on the following page, with the 1st `Fraction` representing onset time, and the second representing duration. It will become apparent that a reconstruction of the data structure is taking place, so it is reasonable to use a new nomenclature for notes, though `N5` and `Note` are functionally equivalent.

Code 7.4: 5-tuplet Note Representation

$$type\ N5\ =\ (Int, Int, Int, Int, Int)$$

The next hierarchic level was the `Crd` element, a list of notes. Although this structure remains, now as a list of `N5` notes, the term *chord* is now more misleading, since the list may represent either a vertical or horizontal structure. To differentiate between the two, two additional homo-types , `HSlice` for horizontal lists and `VSlice` for vertical, are introduced, shown in Code 7.5.

Code 7.5: Redefined Crd and Homo-Types

$$type\ Crd\ \ =[N5]$$
$$type\ HSlice\ =[N5]$$
$$type\ VSlice\ =[N5]$$

At the top level, a similar strategy is employed with two additional homo-types for `Seq`. These contain the `Crd` homo-types which are orthogonally oriented within their list. Thus an `HSlab` is a horizontal list of `VSlice` vertical note groupings. The three top-level structures are described in Code 7.6.

Code 7.6: Redefined Seq and Homo-Types

$$type\ Seq\ \ =[[N5]]$$
$$type\ HSlab\ =[VSlice]$$
$$type\ VSlab\ =[HSlice]$$

Figure 7.6 on the next page is included as an example of an imported denemo file; in this case, the first two bars of the Prelude BWV1007.

Figure 7.6: Prelude BWV1007, 5-tuplet encoding

```
[[(23,-2,4,0,1/16),(6,-1,6,1/16,1/16),(35,-1,8,1/8,1/16),
(29,-1,10,3/16,1/16),(35,-1,13,1/4,1/16),(6,-1,15,5/16,1/16),
(35,-1,17,3/8,1/16),(6,-1,19,7/16,1/16),(23,-2,22,1/2,1/16),
(6,-1,24,9/16,1/16),(35,-1,26,5/8,1/16),(29,-1,28,11/16,1/16),
(35,-1,31,3/4,1/16),(6,-1,33,13/16,1/16),(35,-1,35,7/8,1/16),
(6,-1,37,15/16,1/16),(23,-2,40,1,1/16),(12,-1,42,17/16,1/16),
(0,0,44,9/8,1/16),(35,-1,46,19/16,1/16),(0,0,49,5/4,1/16),
(12,-1,51,21/16,1/16),(0,0,53,11/8,1/16),(12,-1,55,23/16,1/16),
(23,-2,58,3/2,1/16),(12,-1,60,25/16,1/16),(0,0,62,13/8,1/16),
(35,-1,64,27/16,1/16),(0,0,67,7/4,1/16),(12,-1,69,29/16,1/16),
(0,0,71,15/8,1/16),(12,-1,73,31/16,1/16)]]
```

7.4.2 Repercussions

Prior to the reformation of the data structure, the grouping of notes into vertical time-oriented `Crd` elements was performed at the point of importing data from the denemo XML file, and any temporal information was then discarded. This is no-longer necessary since all temporal information is maintained within each note. Denemo presents notes in terms of horizontally oriented voices, and so the path of least resistance is to accept this as the initial data format within the system i.e. a `VSlab` of `HSlice` elements.

Two functions can now be introduced that transform a `VSlab` into an `HSlab`, according two the first two strategies in Section 7.3.

7.5 Modified Top-Level Combinator

An alternative to `mlex` introduced in Section 6.2.2 on page 100 is shown which does not use predication. In fact, no outcomes are discarded, which exponentially increases the volume of results. Firstly part of the `mlex` combinator is

abstracted out. Since it is common to both top-level combinators it seems logical to code it only once, as well as making the code of the combinators less obtuse.

Code 7.7: Abstracted Function

```
op              :: (Parser Seq, String) → Parser Token → Parser Token
(p, t) `op` xs =(p `tok` t) `alt` xs
```

The combinator itself branches at every point where more than one structure is present at the head of the input. The helper function `tokenjoin` is added to combine the results and recurse to the `combiner` function. Examining the functions in Code 7.8, it can be seen that the need for including the `seqitem` parser as an element of the list `l` of parsers is obviated. Failure of all the parsers in `l` simply invokes the combinator again beginning with the next element of the input.

Code 7.8: Top-level Combinator

```
combiner :: [(Parser Seq, String)] → Parser [Token]
combiner l inp
  | inp == [] = [([("End", inp)], inp)]
  | otherwise = case (foldr op never l inp) of
                  [] → combiner l (tail inp)
                  x → concat [tokenjoin (t, s, l) | (t, s) ← x]


tokenjoin          :: (Token, Seq, [(Parser Seq, String)]) → [([Token, Seq)]
tokenjoin (t, s, l) =[(t : t', s') | (t', s') ← (combiner l s)]
```

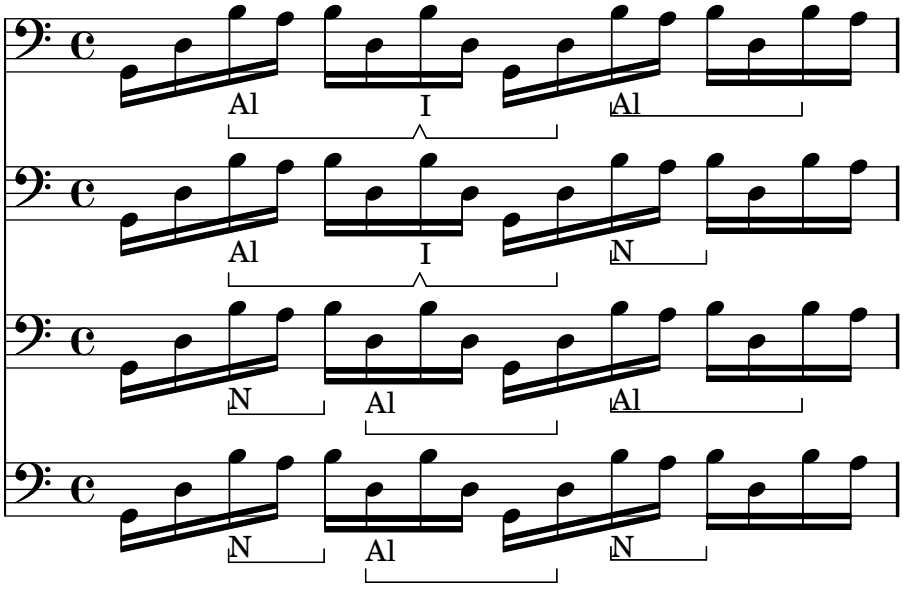
The effect of this parser is to create a new entry in the list of results whenever for any non-singular result at a given point int he input. This causes a large volume of data to be produced, but it has the advantage of producing every potential parse of the input explicitly. Storing and accessing this data is an area where considerable time and computational effort could be saved, but it is

the nature of the results which is of interest and this is not materially impacted by the system as it stands.

7.5.1 Using the New Combinator

Figure 7.7 shows the new top-level combinator in use. The results at the command line are abbreviated for clarity but it can be seen that there are 4 sets of results. These are produced by two ambiguities, each with two possible interpretations. It can be seen that the total number of interpretations is the product of the ambiguities at each juncture. Due to the structure of the first four bars of this Prelude (BWV1007), each bar containing two bi-ambiguities, there are 256 possible interpretations of those first four bars alone. Given that there are at least 20 bi-ambiguities in the Prelude, analysing it in it's entirety may not be practical.

Figure 7.7: Using the new Combinator




```
 ~$ combiner1 [(pedal, "Pdl"),(stepwise, "St"),(neighbour, "Nb"),
((interesting [pedal,stepwise,neighbour]),"Ig")]
[[(23,-2,4,0,1/16)], ....,
[(6,-1,37,15/16,1/16)]])

[([("Pdl",[[(35,-1,8,1/8,1/16)],....,
[(35,-1,35,7/8,1/16)]]),("End",[])],[]),
([("Pdl",[[(35,-1,8,1/8,1/16)],....,
[(35,-1,31,3/4,1/16)]]),("End",[])],[]),
([("Nb",[[(35,-1,8,1/8,1/16)],....,
[(35,-1,35,7/8,1/16)]]),("End",[])],[]),
([("Nb",[[(35,-1,8,1/8,1/16)],....,
[(35,-1,31,3/4,1/16)]]),("End",[])],[])]
```

Chapter 8

Results

To assess the usefulness of the results produced by the system, they must be compared to human-prepared analyses of the same pieces of music. The Diran Alexanian edition of the Bach 'cello suites [12] is a richly annotated performance edition, developed by a cellist who enjoyed a decades long collaboration with Cassals. This 1929 edition makes explicit the intended performance structures, as envisaged by the editor. Taking as an example the first twelve bars of the first Gavotte from the 5th suite a comparison can be made between the Alexanian edition in Figure 8.1 on the following page and the popular student edition of August Wenzinger [13] in Figure 8.2 on page 123.

There are, most significantly, differences in the actual pitch content between these, and other, cello suite editions. This is a consequence of the lack of an autograph edition, all good editions being based on a small number of historical transcriptions. A facsimile of the Anna Magdalena Bach is included with the Alexanian edition, although the accuracy of it's transcription is frequently

Figure 8.1: Alexanian edition (Salabert, 1929)

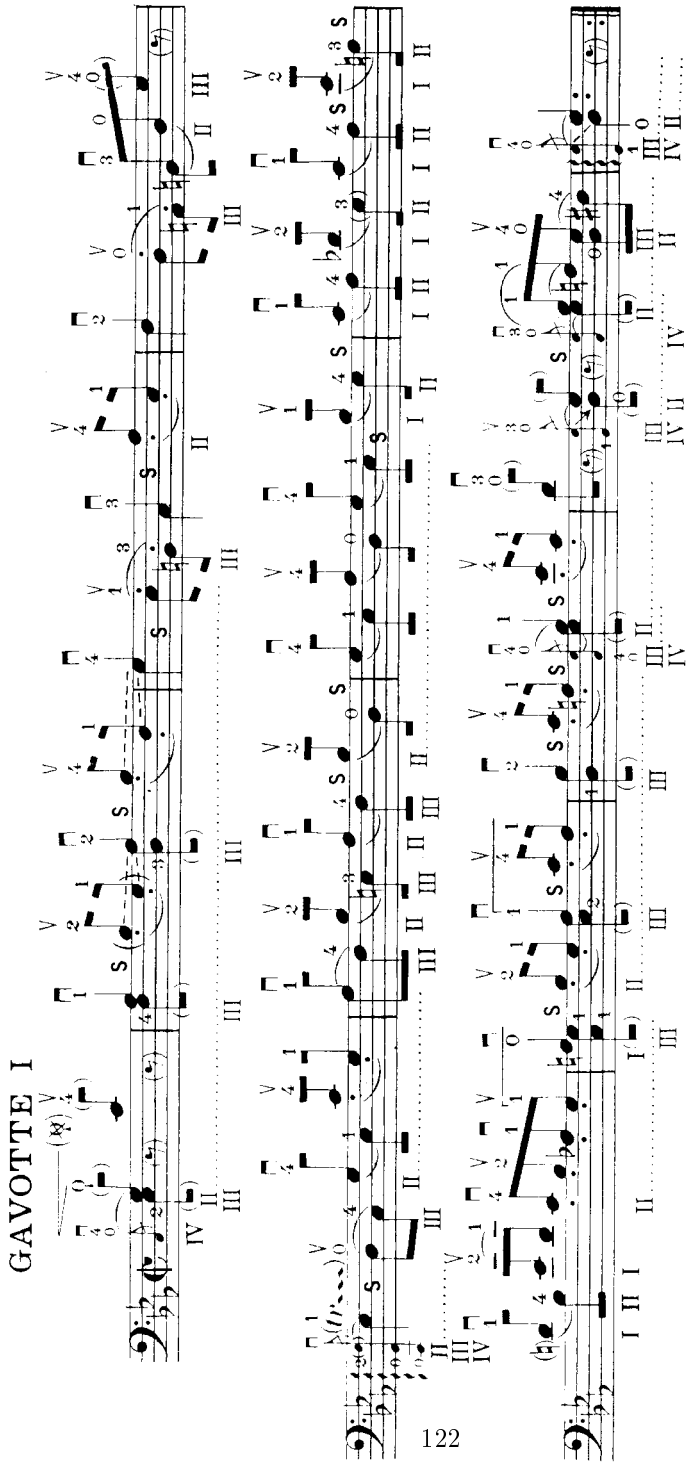
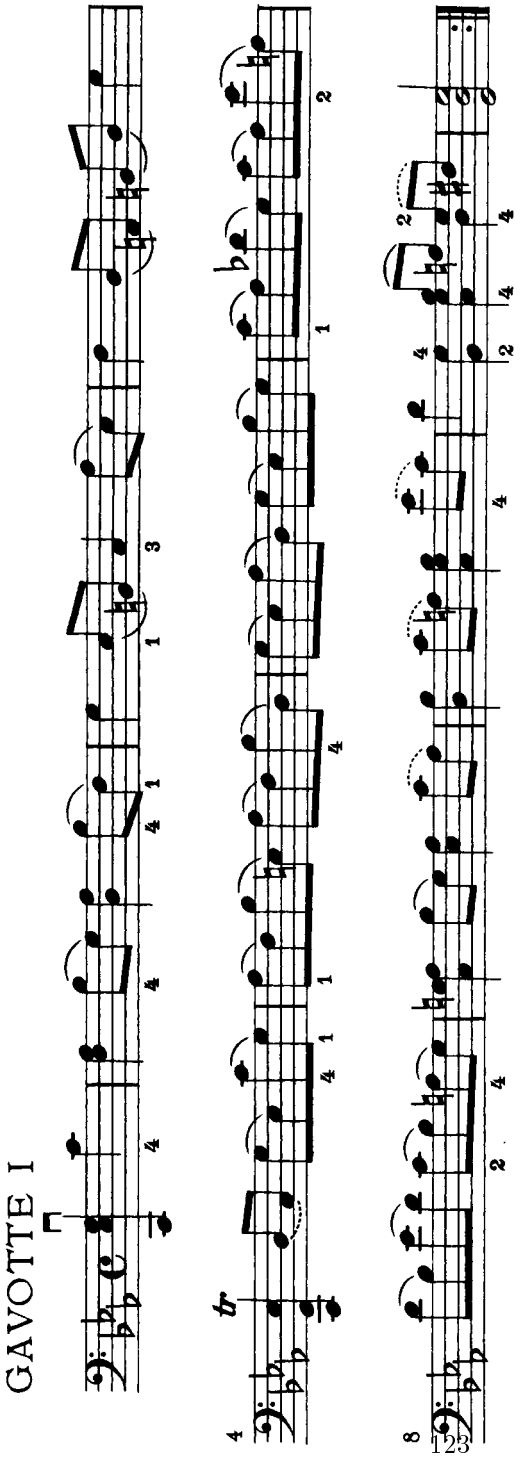


Figure 8.2: Wenzinger edition (Bärenreiter, 1950)

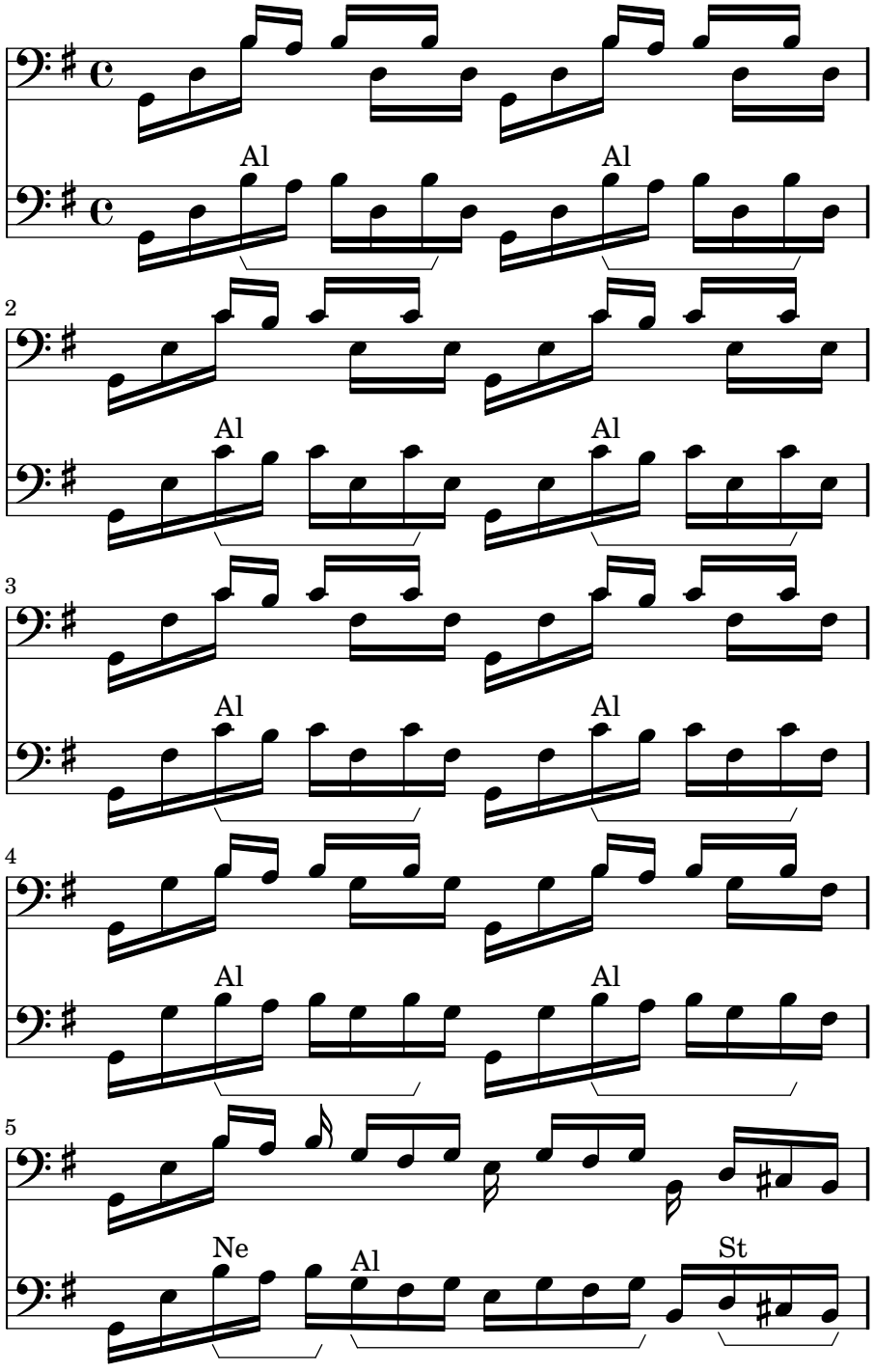


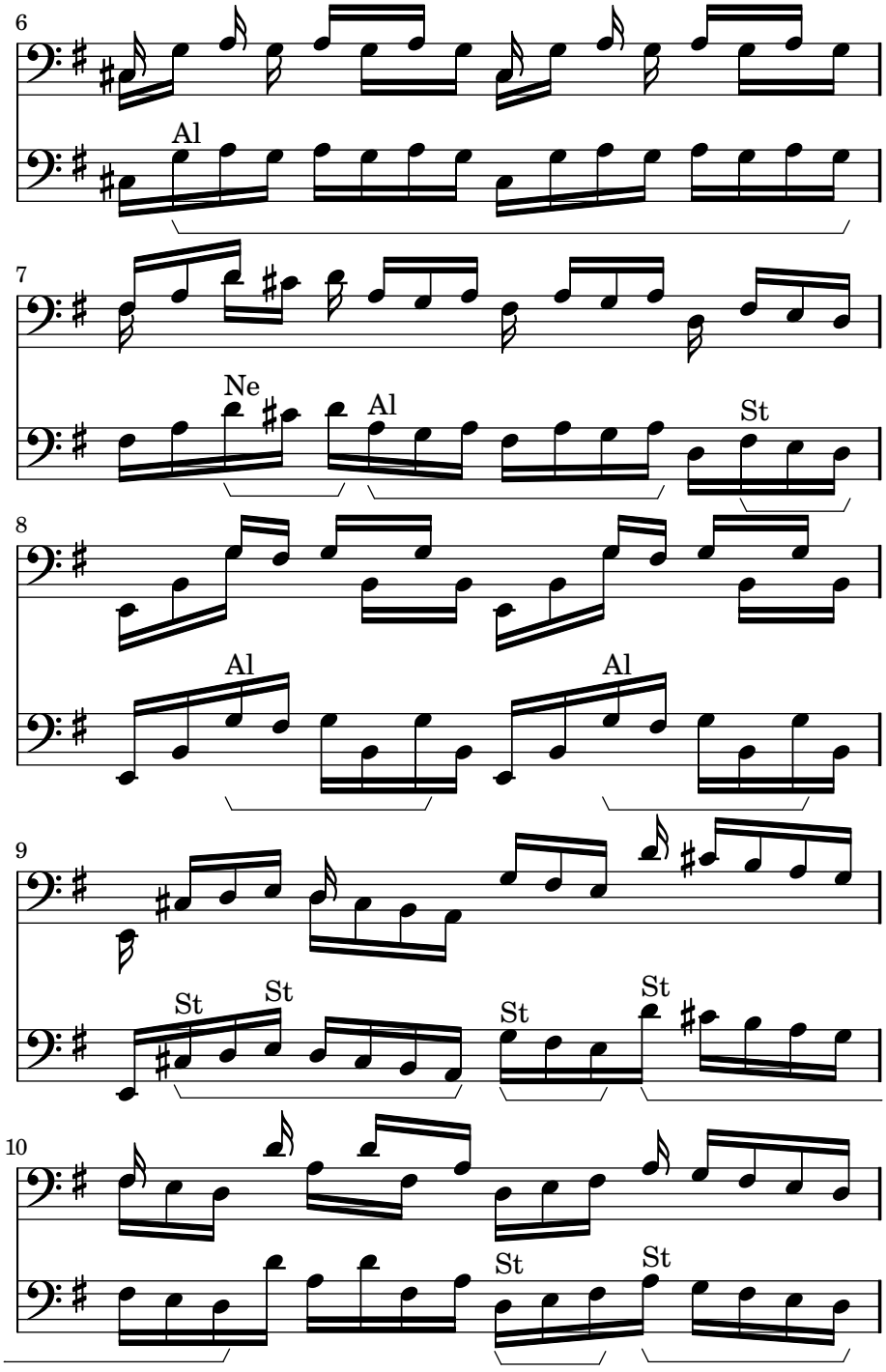
questioned, and there are certainly inconsistencies, which are generally taken as errors. The other two named sources being a copy by J P Kellner and another by J C Westphal.

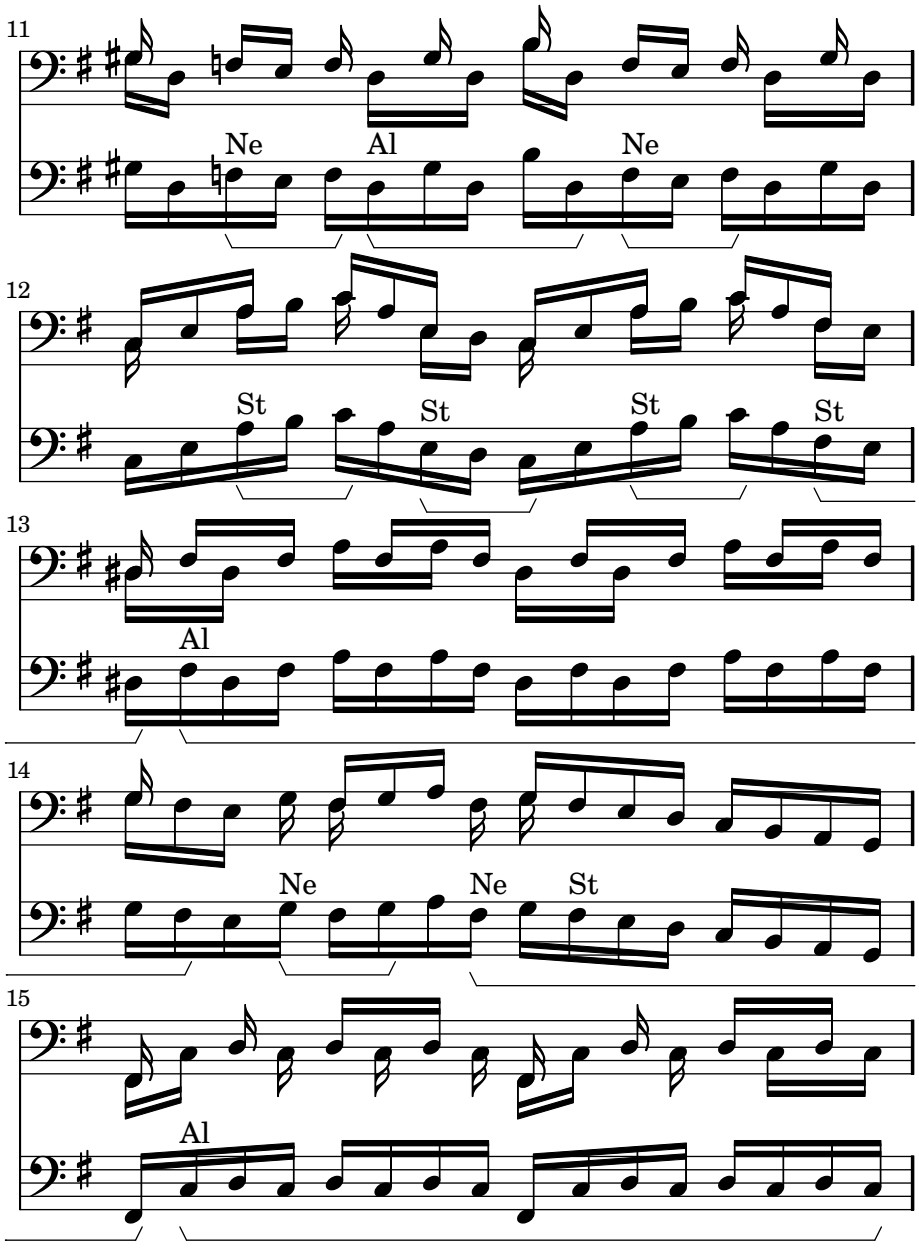
The Alexanian edition can be seen to have a much more explicit fingering notation than the Wenzinger edition. Every new note following a change of position is explicitly fingered with the intended string written beneath the stave in roman numerals. The Wenzinger fingering is sparse by comparison, only indicating a fingering at an (implied) change of position.

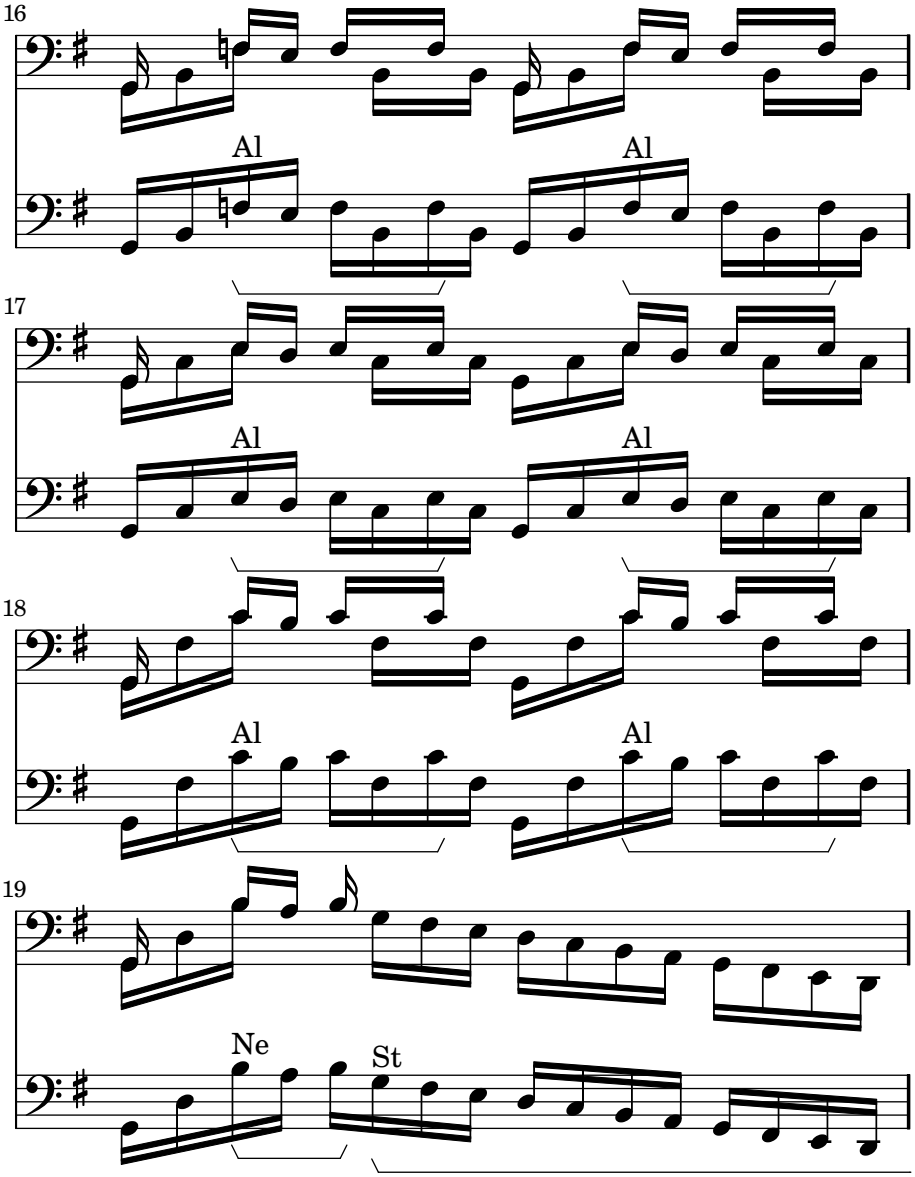
8.1 Simple Combination

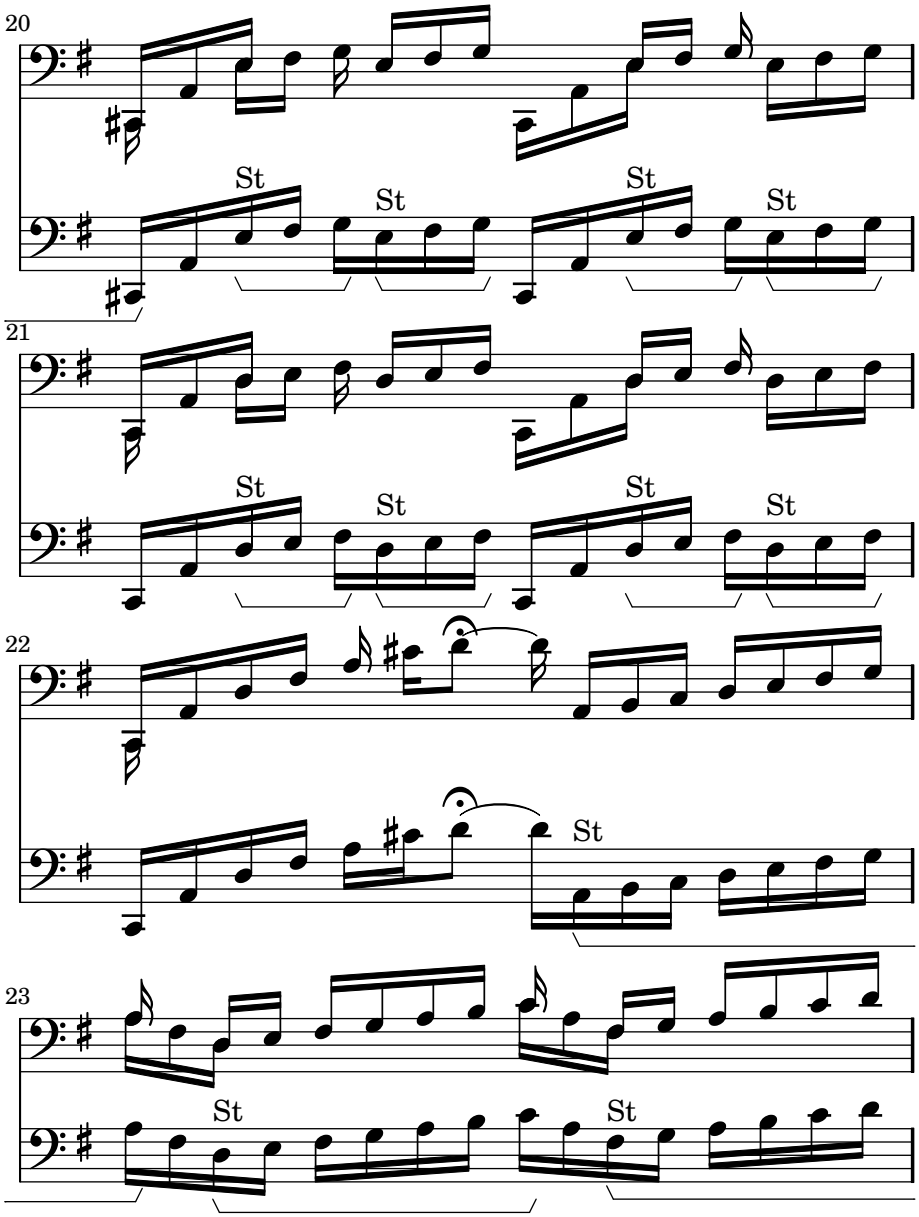
The first full analysis was performed with the system as at Section 6.2.2 on page 100. The notated structures are alternations, stepwise movements, and neighbour motifs, notated `Al`, `St`, and `Ne` respectively. Because the preferential combinator `mlex` was used for this analysis, this is also the precedence order for discovered structures. The beaming instructions from the Alexanian edition are included on the upper staff, and the system results on the lower staff. The bracketing and highlighting of the lower staff are entirely the product of the parser system, and I include an entire movement, rather than cherry-pick a few bars of particularly interesting output.

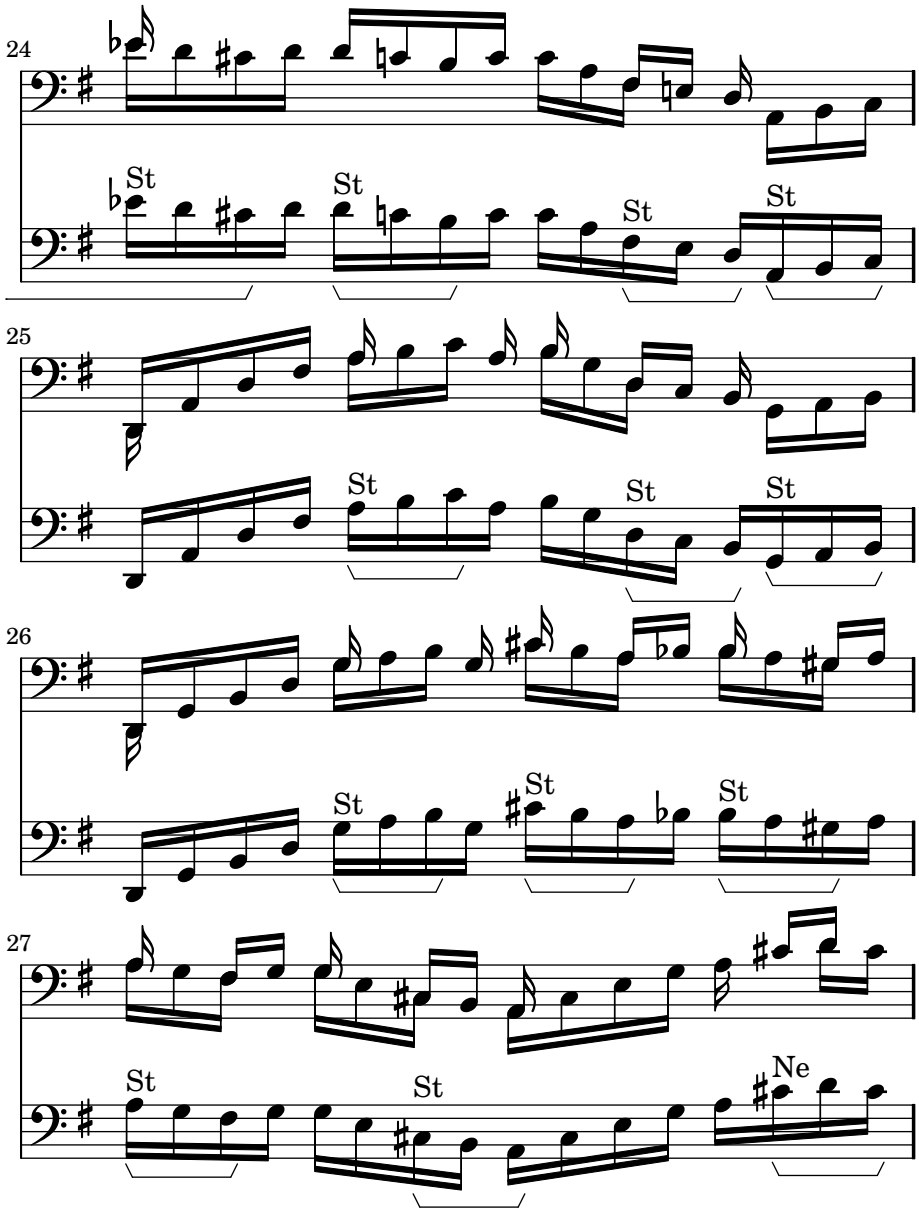

Al
Al
2
Al
Al
3
Al
Al
4
Al
Al
5
Ne
Al
St

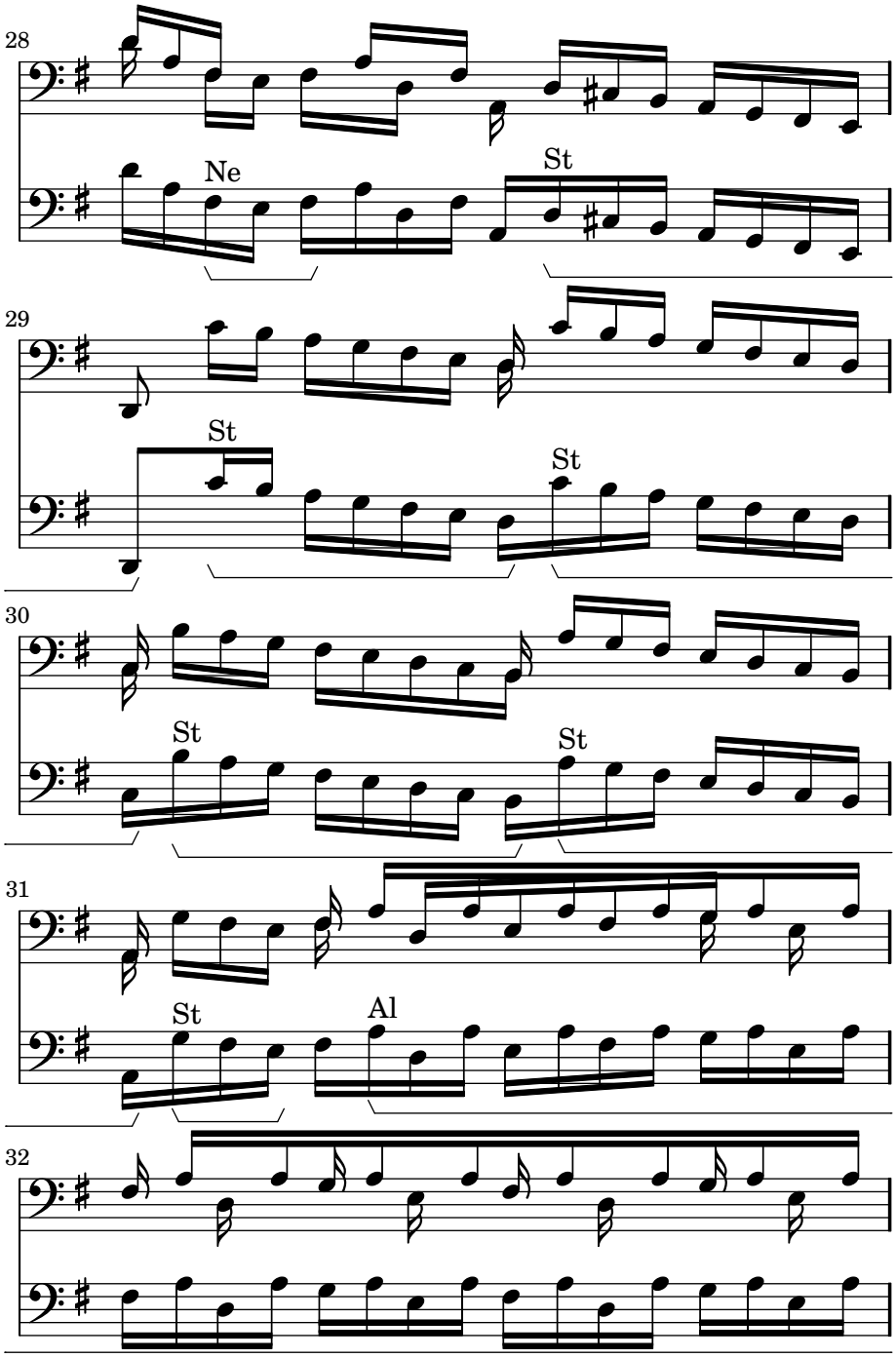

6
Al
7
Ne
Al
St
8
Al
Al
9
St
St
St
St
10
St
St

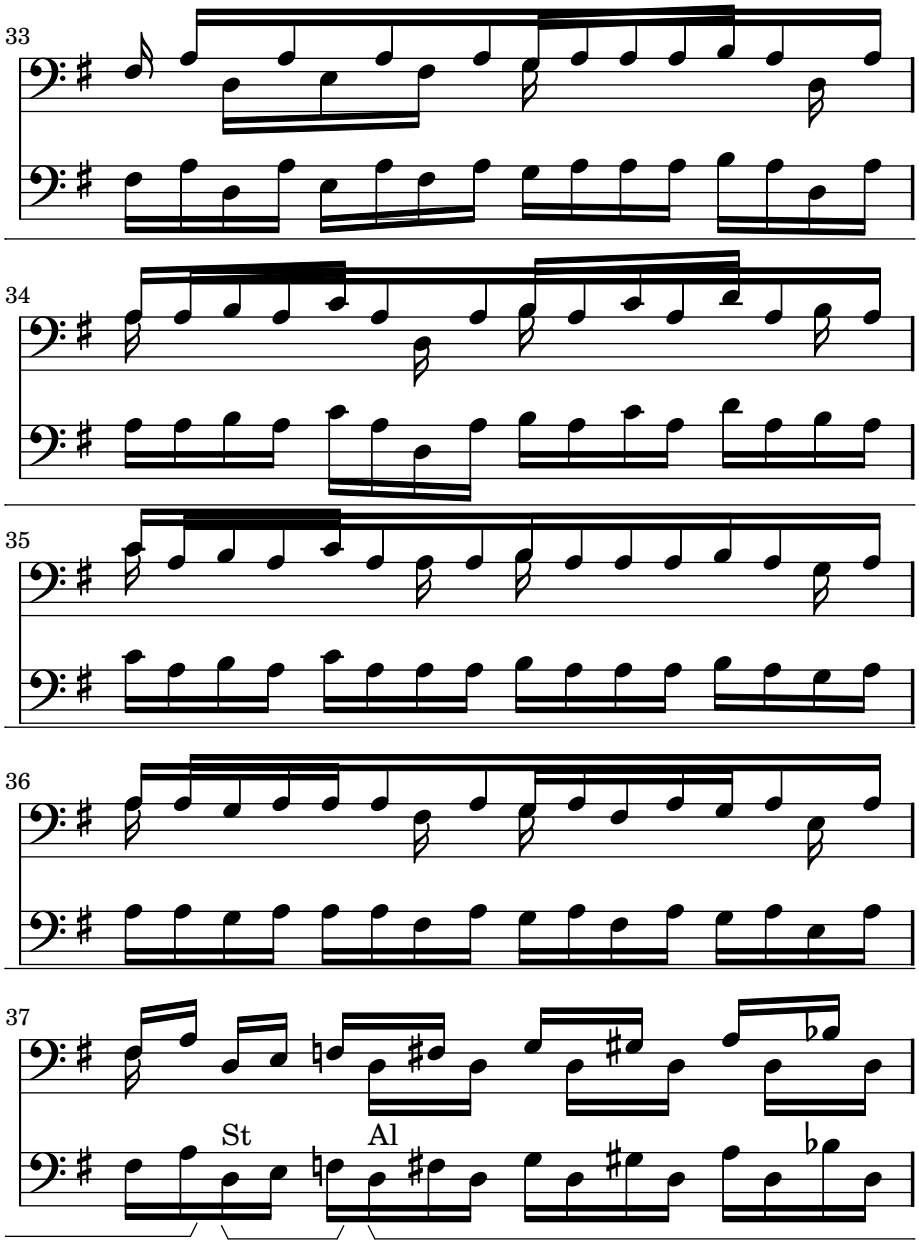

11
Ne
Al
Ne
12
St
St
St
St
13
Al
14
Ne
Ne
St
15
Al

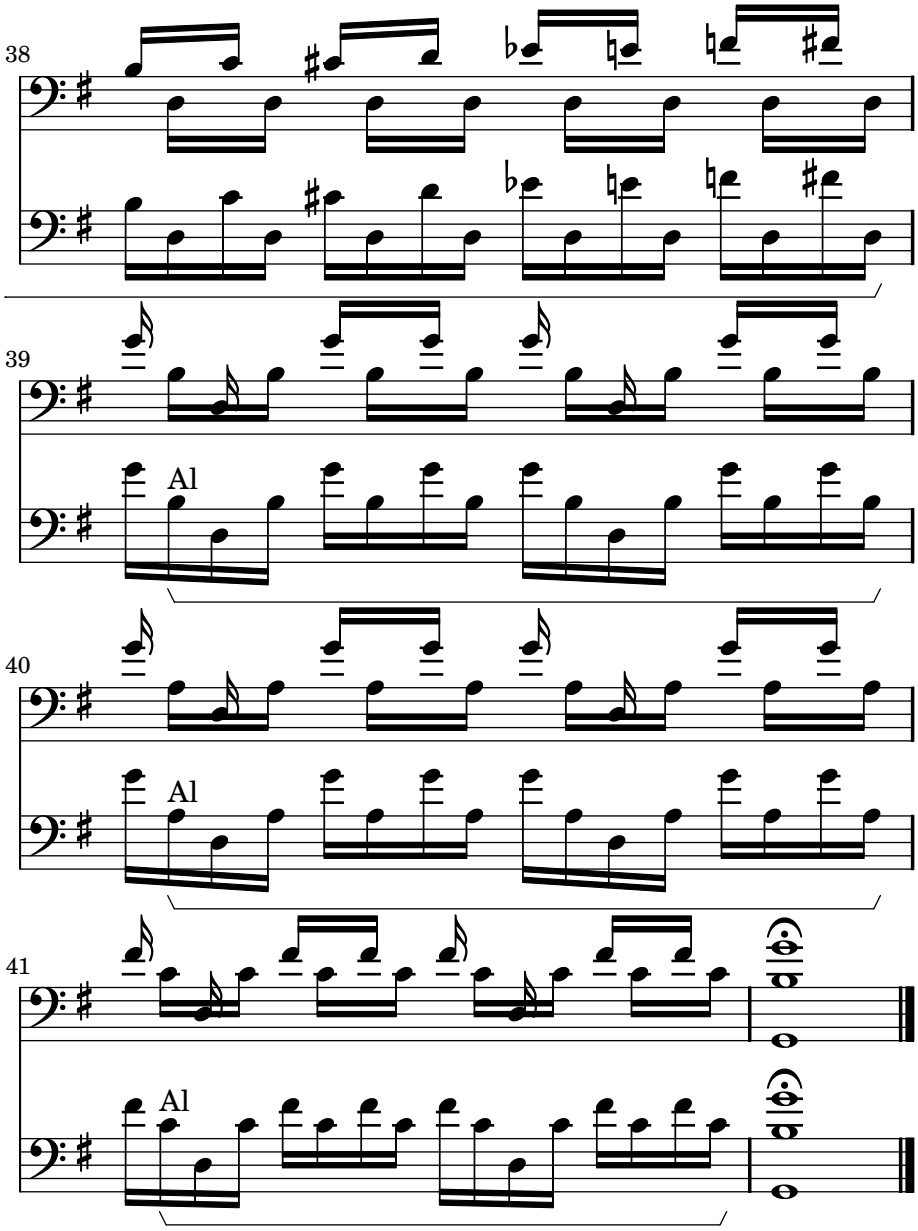

16
Al
Al
17
Al
Al
18
Al
Al
19
Ne
St


20
St
St
St
St
21
St
St
St
St
22
St
23
St
St


24
St
St
St
St
25
St
St
St
26
St
St
St
27
St
St
Ne


28
Ne
St
29
St
St
30
St
St
31
St
Al
32


33
34
35
36
37
St
Al



The system finds two alternation structures in each of the first four bars, which align with structural boundaries in the upper staff. Alexanians structure draws

out the lower and higher notes into two voices. Since no advice is given by the system on how to perform an alternating structure, highlighting the overall figure can be considered a success within the constraints of the system. The neighbour note formed between the 3rd and 5th notes of each half-bar are not detected because of the precedence of the alternation structure; it will be discovered, but only the first structure will be notated.

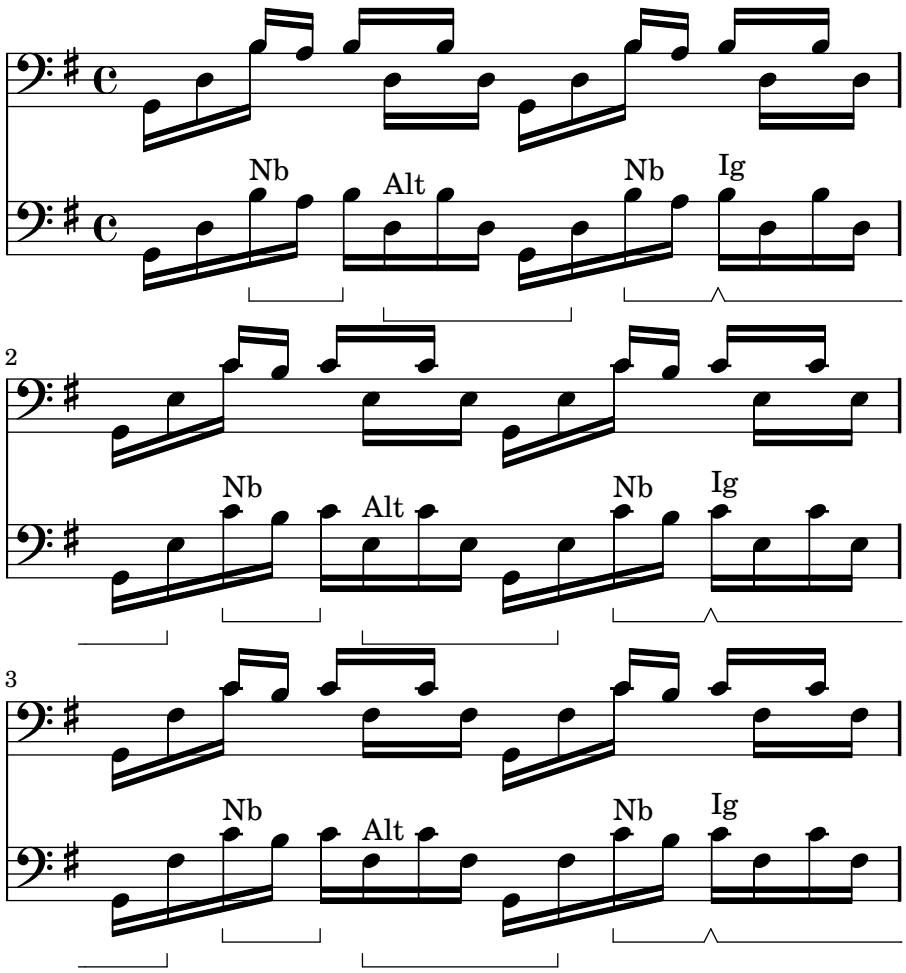
It can be seen in fact that many of the alternation structures discovered by the system contain neighbour structures across 3 pitches. There are two potential resolutions to this problem, either to allow both possibilities simultaneously as in later versions of the system, or to allow the presence of structures within structures, which is not considered in this thesis .

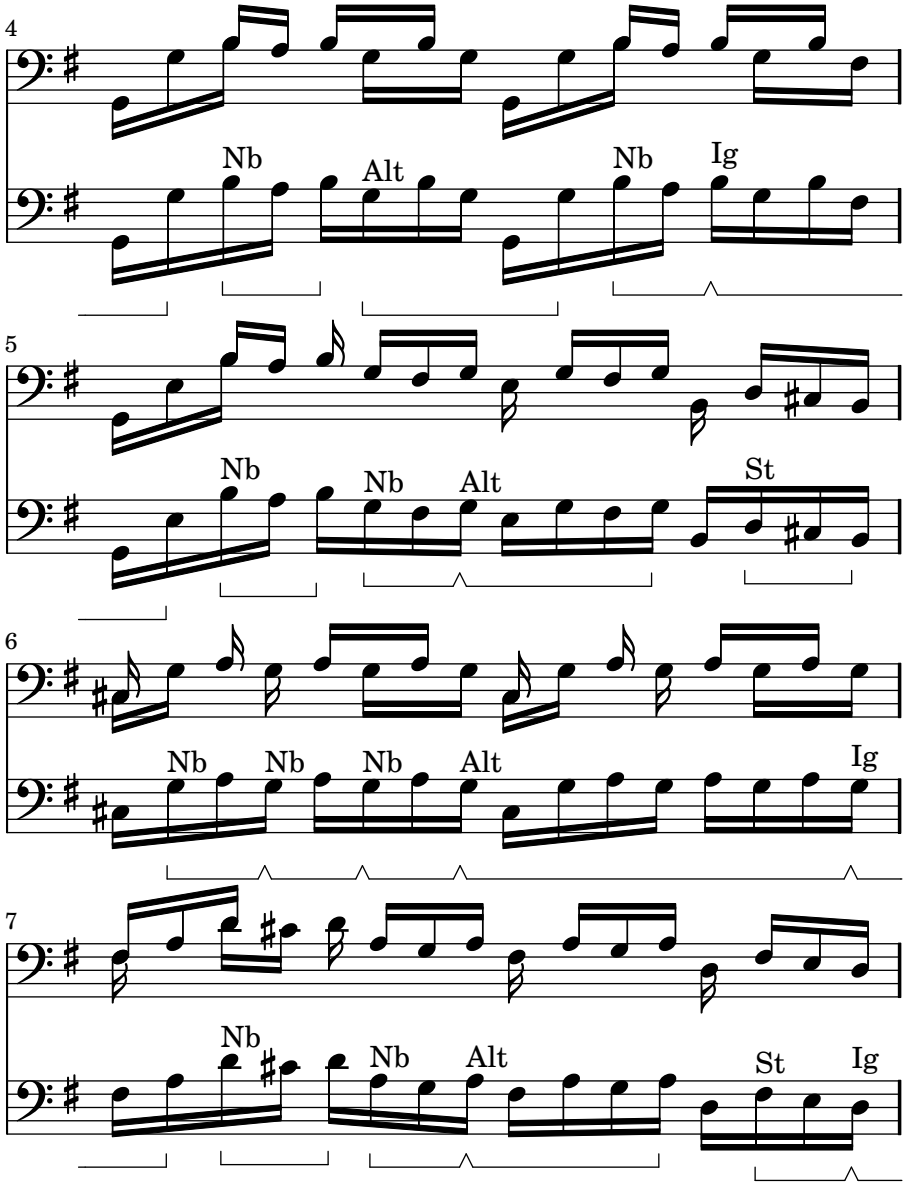
Another artefact within this system can be seen in bar 9, where two stepwise structures have run together. Due to both structures containing the 4th semi-quaver *E*, and as a consequence of using the lilypond analysis brackets, a single bracket spans both structures, accidentally creating a higher level structure. Since this is not an intended result, and the lowest level information is occluded, this is not a desirable visual outcome. The grouping within the intermediate (xml) stages of the results is correct, and so the remedy is outwith the direct scope of this work.

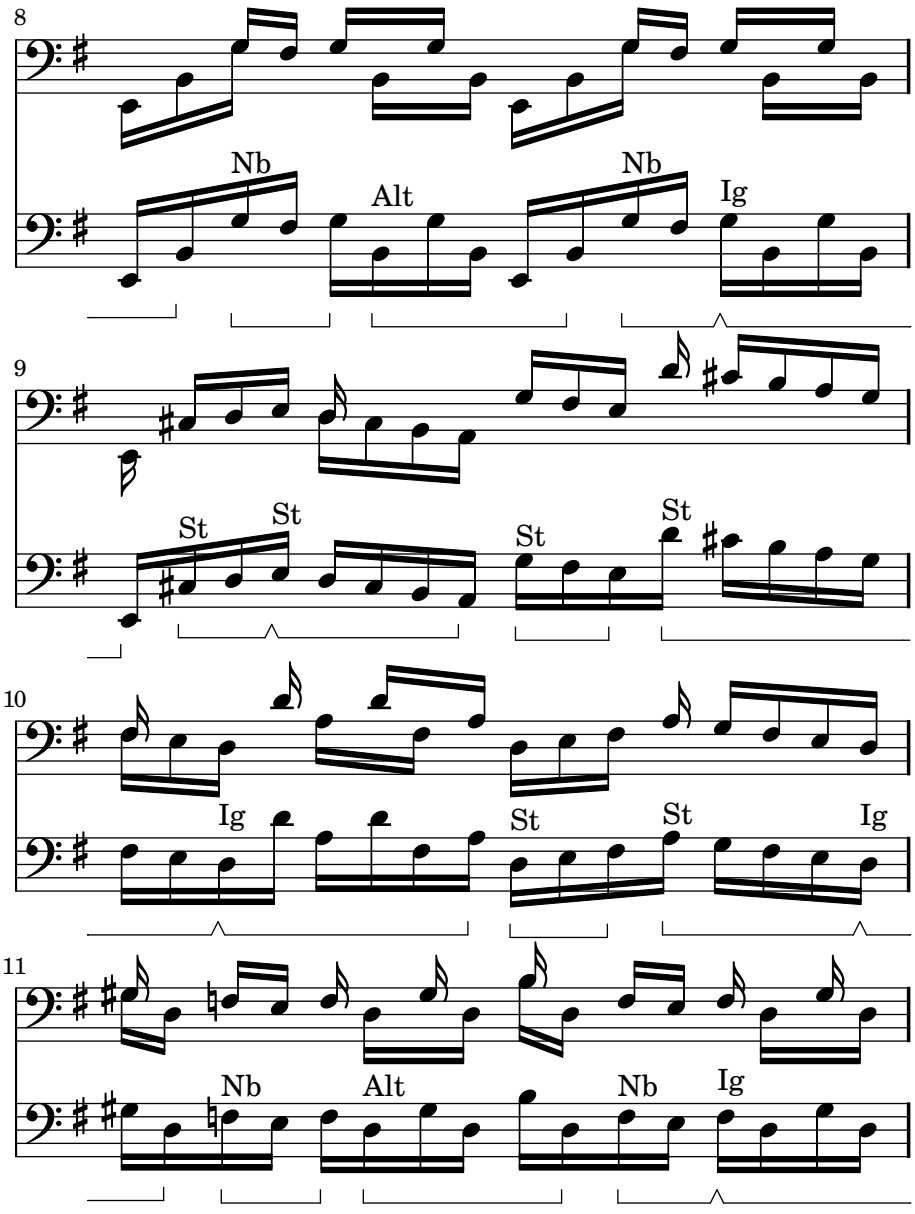
It can also be seen that there is no issue detecting either long structures (e.g. Alternation in bar 13) or bar line crossing (bars 31-36). This was not expected, but validates the systems handling of the input data.

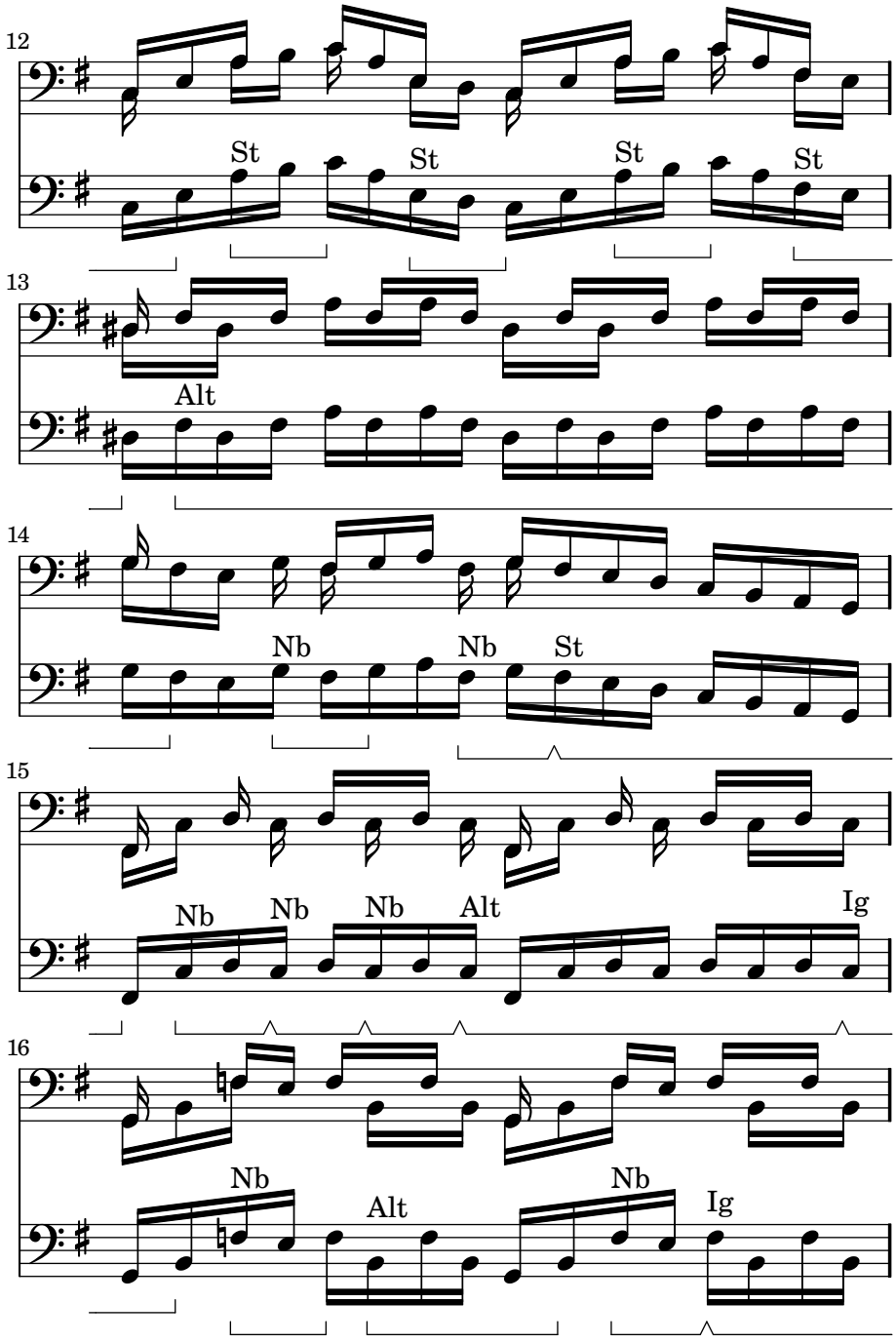
8.2 Alternative Simple Combination

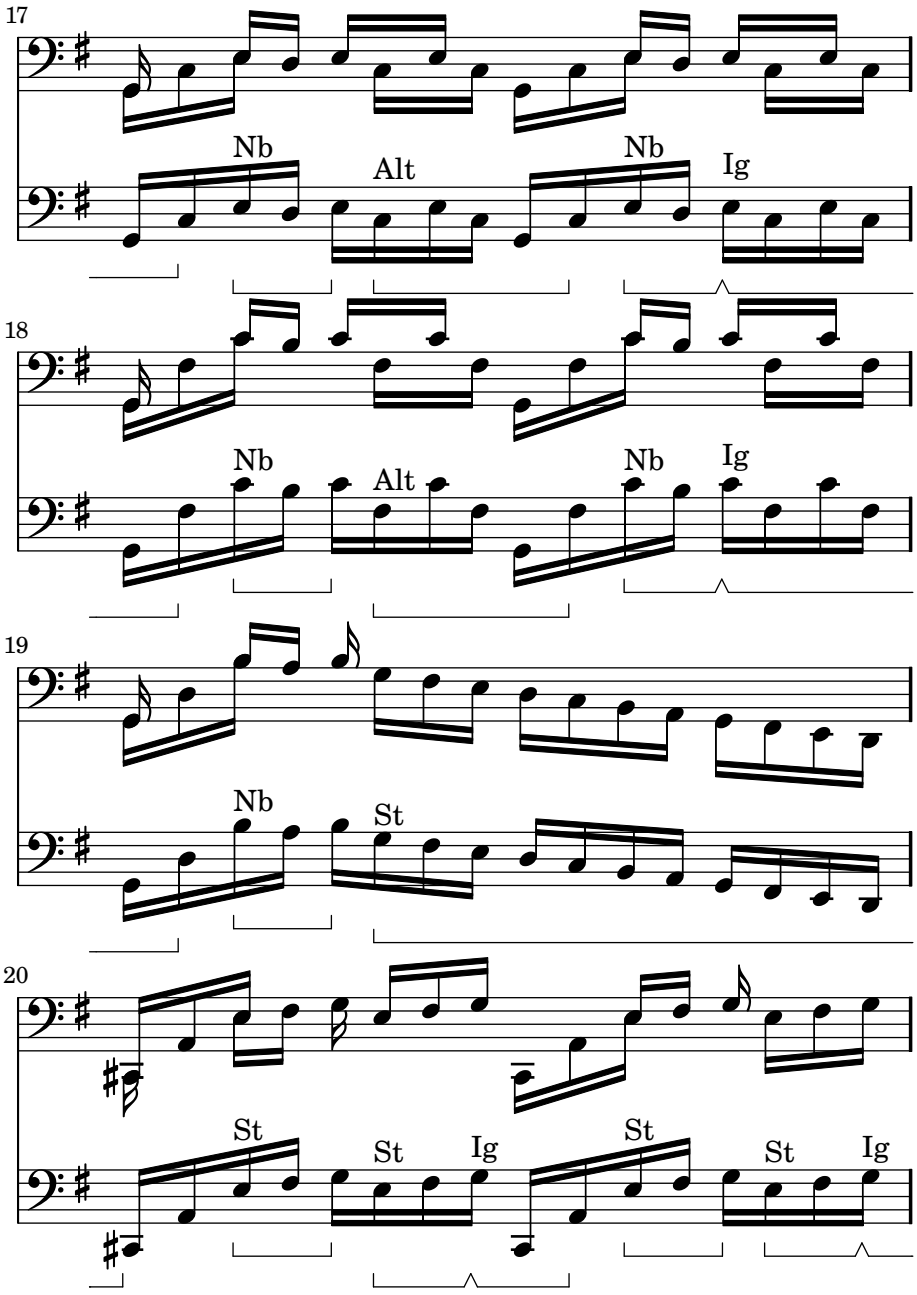
In the analysis below, the order of the parsers supplied to the `mlex` combinator is reversed, i.e. neighbour, stepwise, alternation. The parser for interesting structures is also included, based on the failure of the other three named parsers. This analysis is annotated using the lilypond pedal brackets.

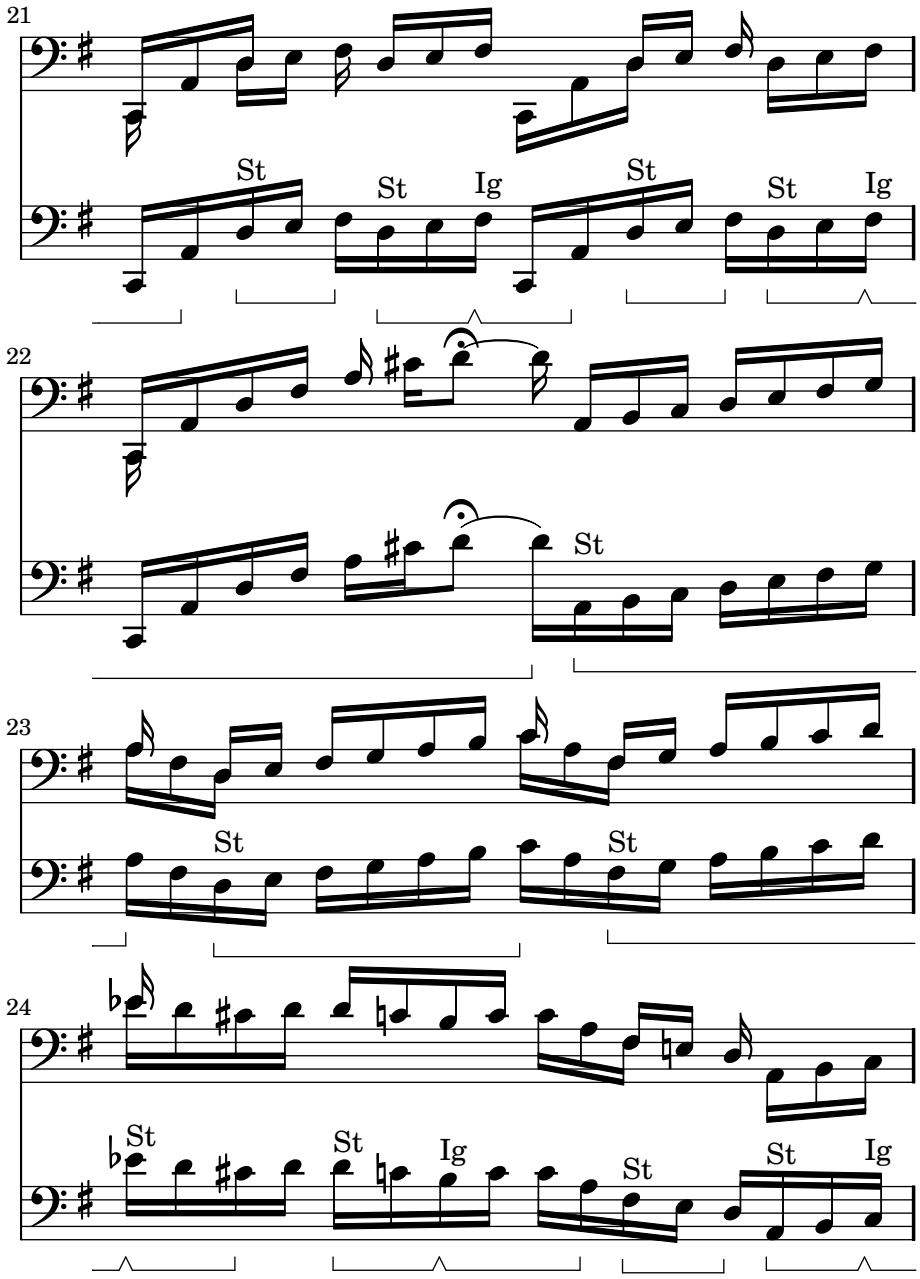


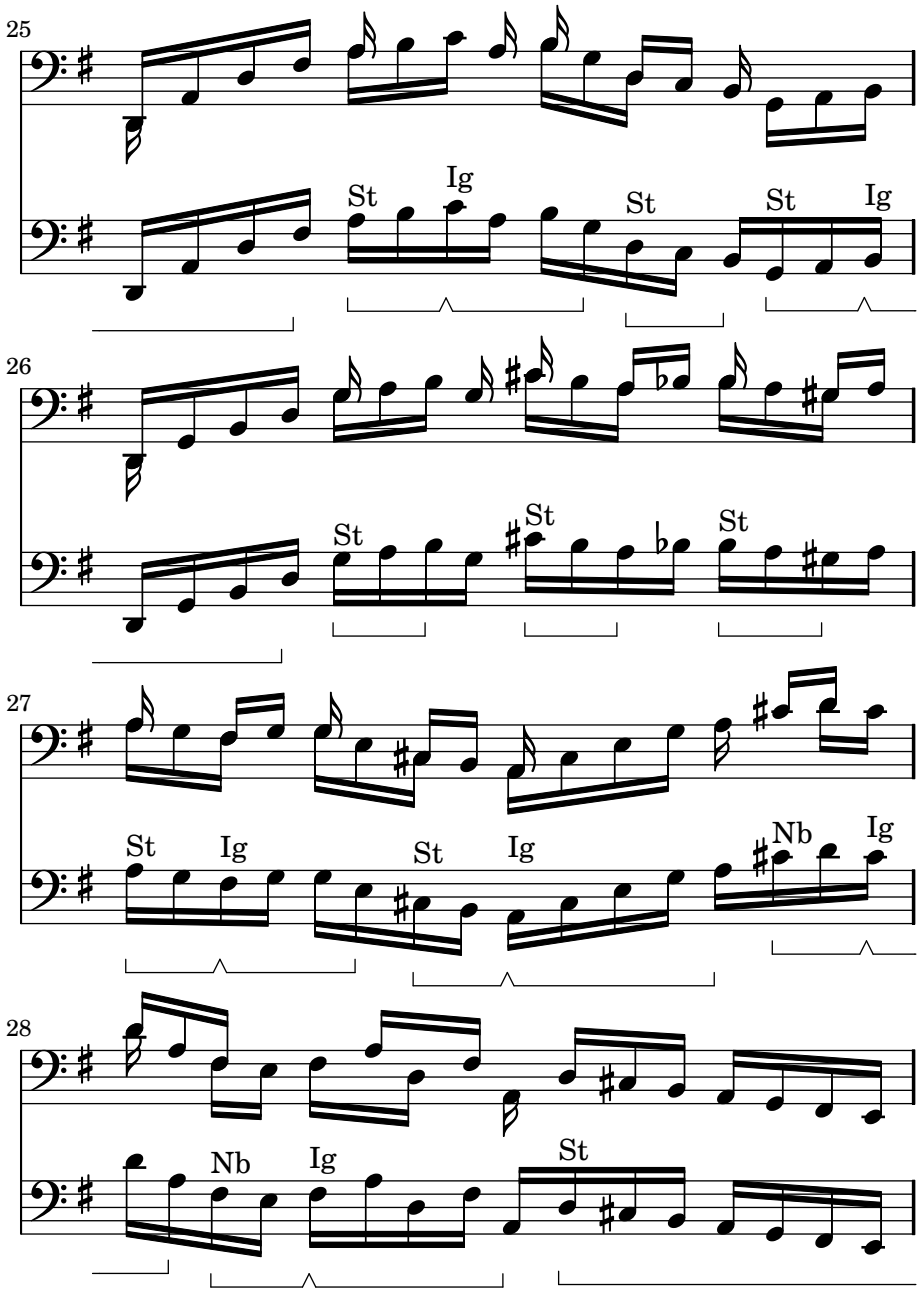

4
Nb
Alt
Nb
Ig
5
Nb
Nb
Alt
St
6
Nb
Nb
Nb
Alt
Ig
7
Nb
Nb
Alt
St
Ig

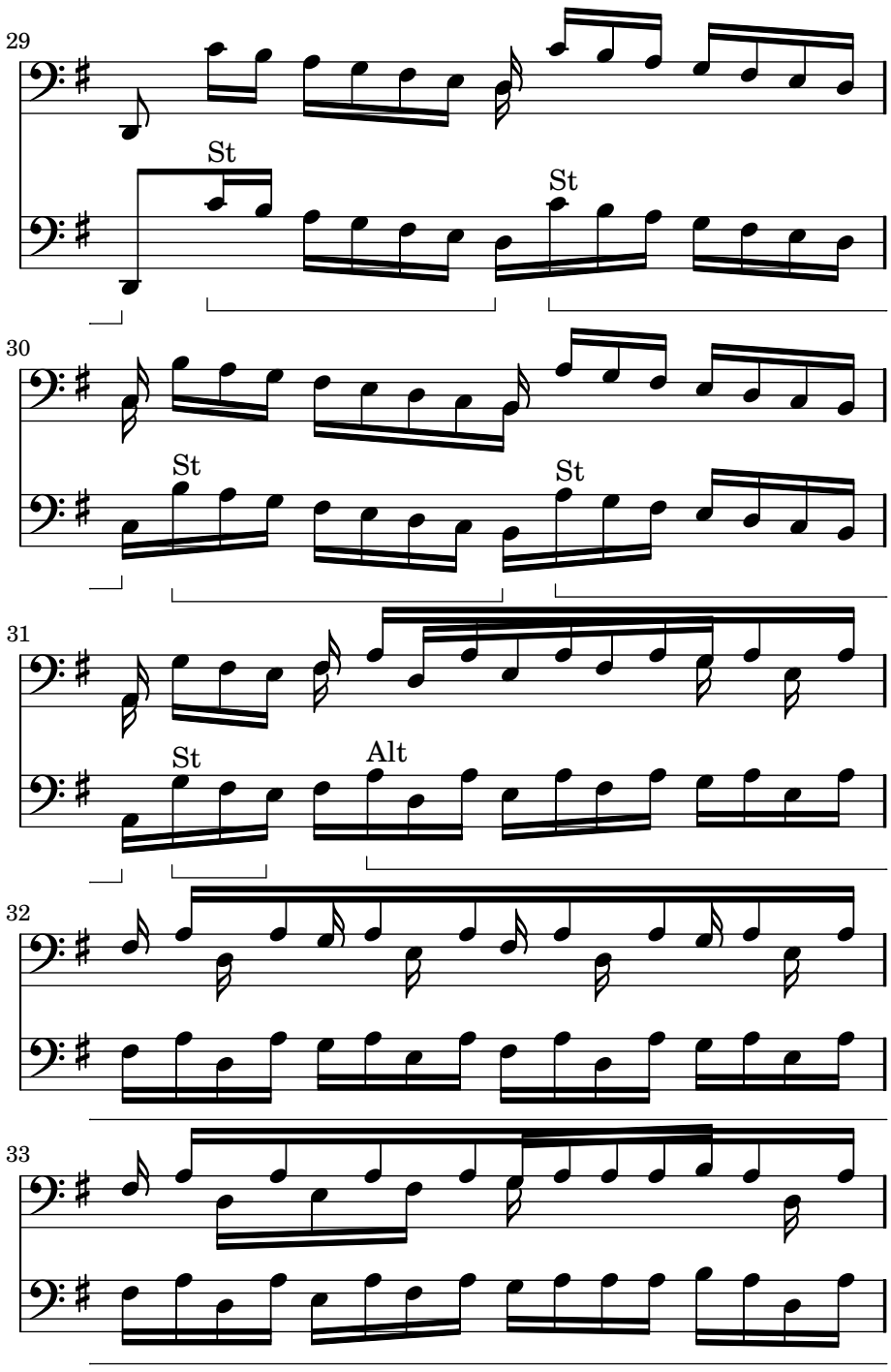

8
Nb
Alt
Nb
Ig
9
St
St
St
St
10
Ig
St
St
Ig
11
Nb
Alt
Nb
Ig

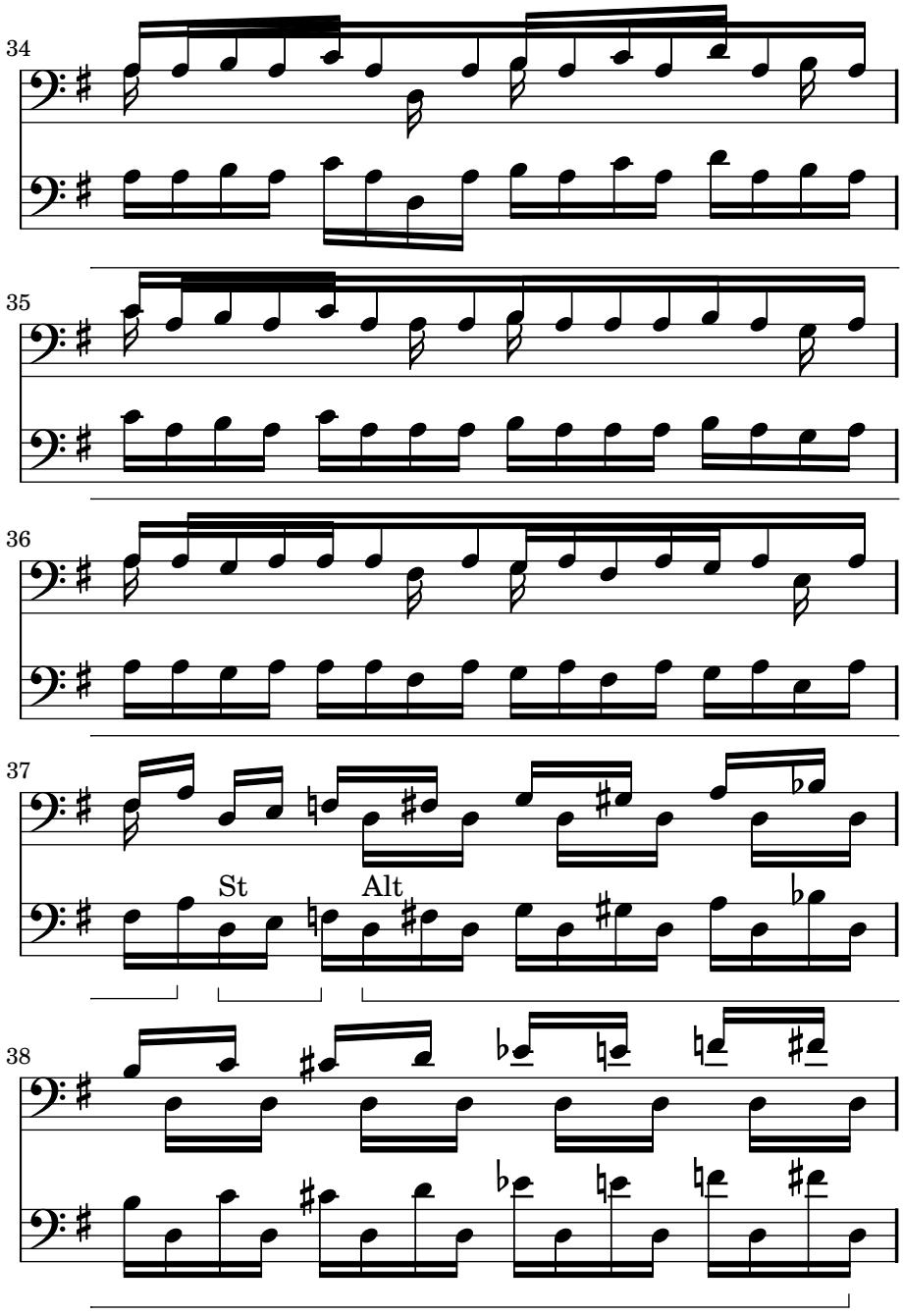

12
St
St
St
St
13
Alt
14
Nb
Nb
St
15
Nb
Nb
Nb
Alt
Ig
16
Nb
Alt
Nb
Ig

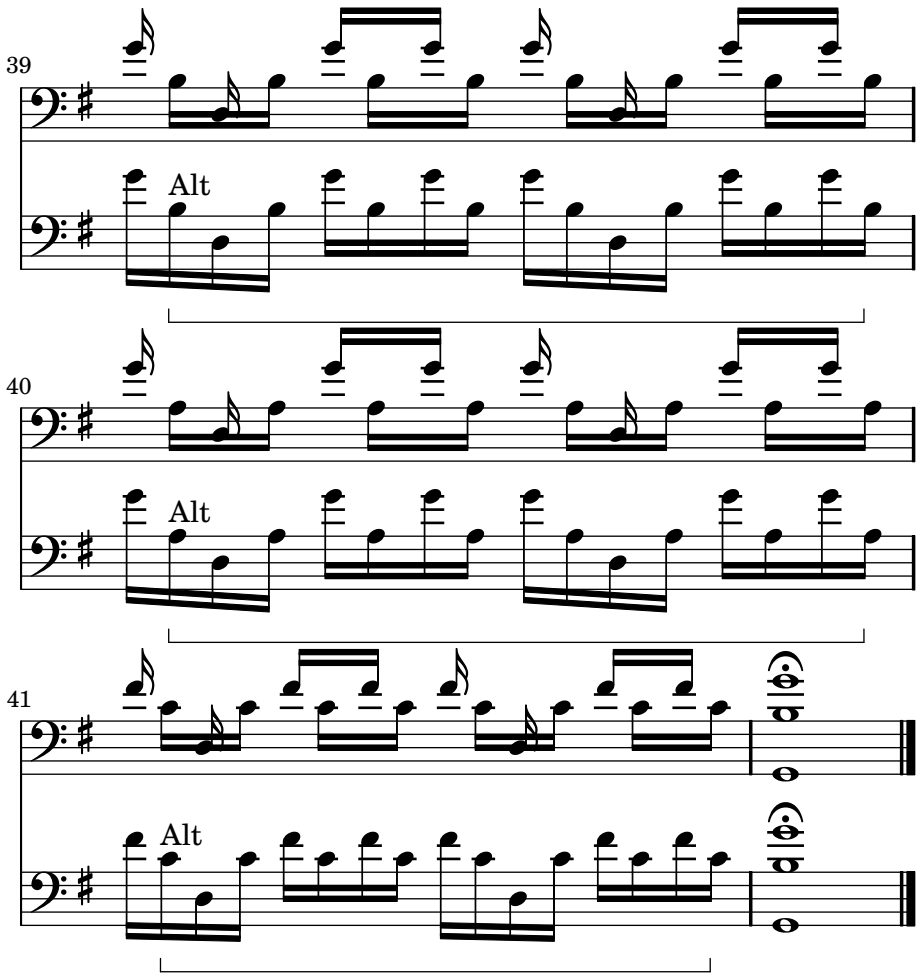

17
Nb
Alt
Nb
Ig
18
Nb
Alt
Nb
Ig
19
Nb
St
20
St
St
Ig
St
St
Ig


21
St
St
Ig
St
St
Ig
22
St
23
St
St
24
St
St
Ig
St
St
Ig


25
St
Ig
St
St
Ig
26
St
St
St
27
St
Ig
St
Ig
Nb
Ig
28
Nb
Ig
St


29
St
St
30
St
St
31
St
Alt
32
33


34
35
36
37
St
Alt
38



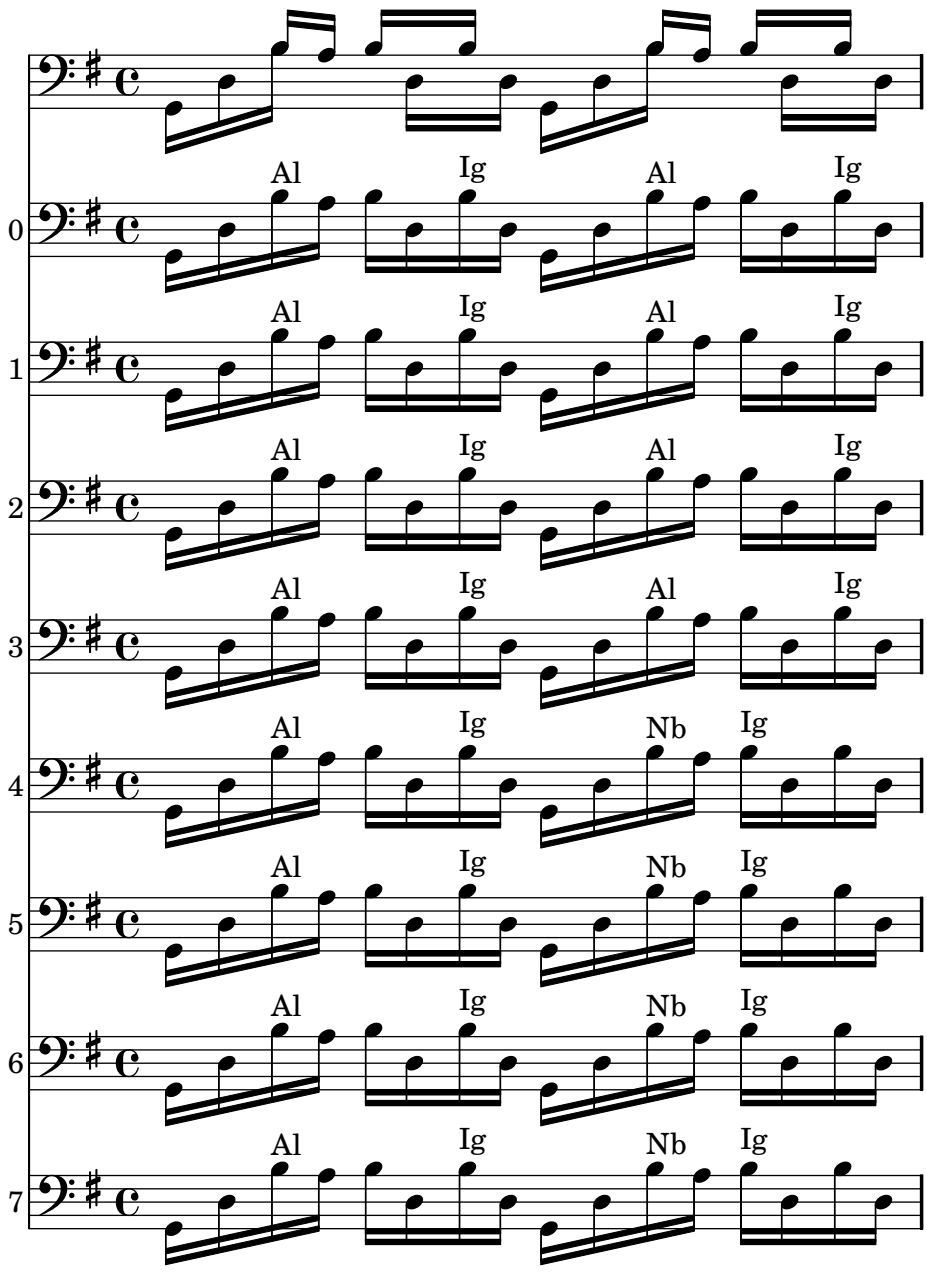
The principle differences in the results are the detection of neighbour structures where alternation structures had occurred. This has a knock on effect of detecting further new structures previously obscured by the alternation structures.

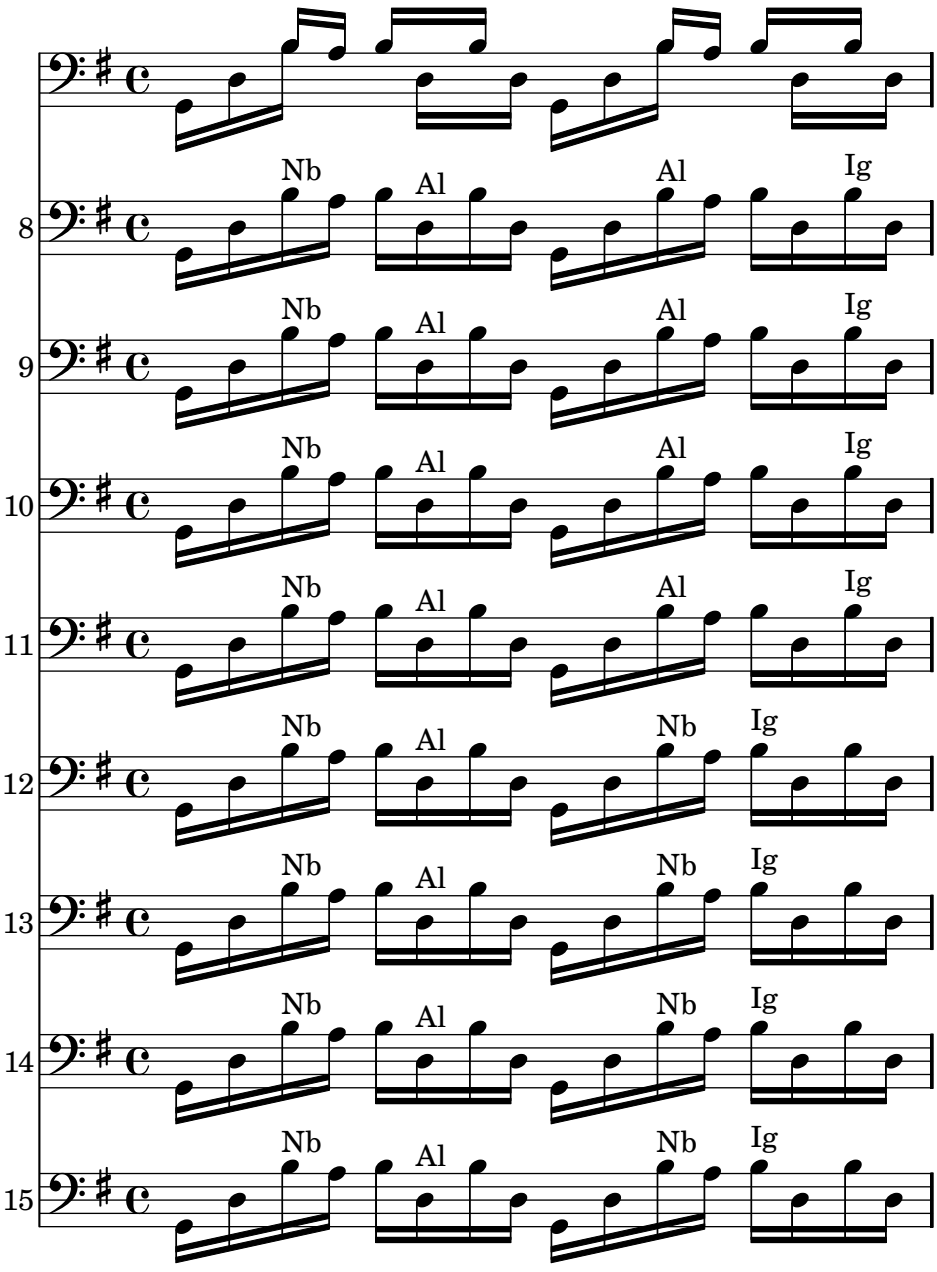
If we compare the first bar from each analysis, the neighbour feature is now correctly identified, and would match up with the performance edition, while the alternation structure is less helpful. We can also find several Ig results in this reading, relating to potentially interesting results. Some of these turn out

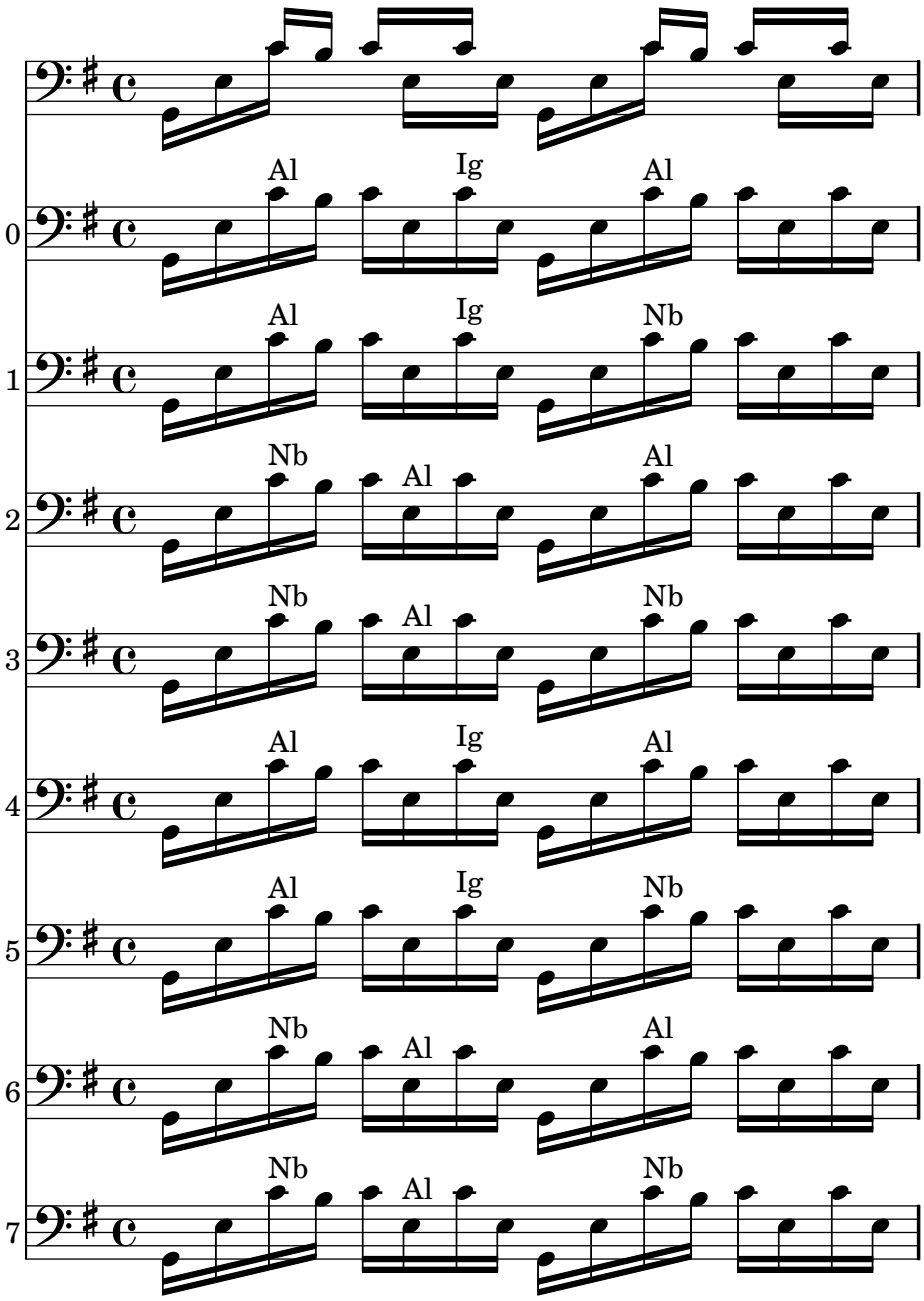
to not be terribly interesting, but we can see in bars 25,27 & 28 structures which would be described as being broken chords, or arpeggio based. For the analyst-developer, these would be pointers to help design new micro parsers.

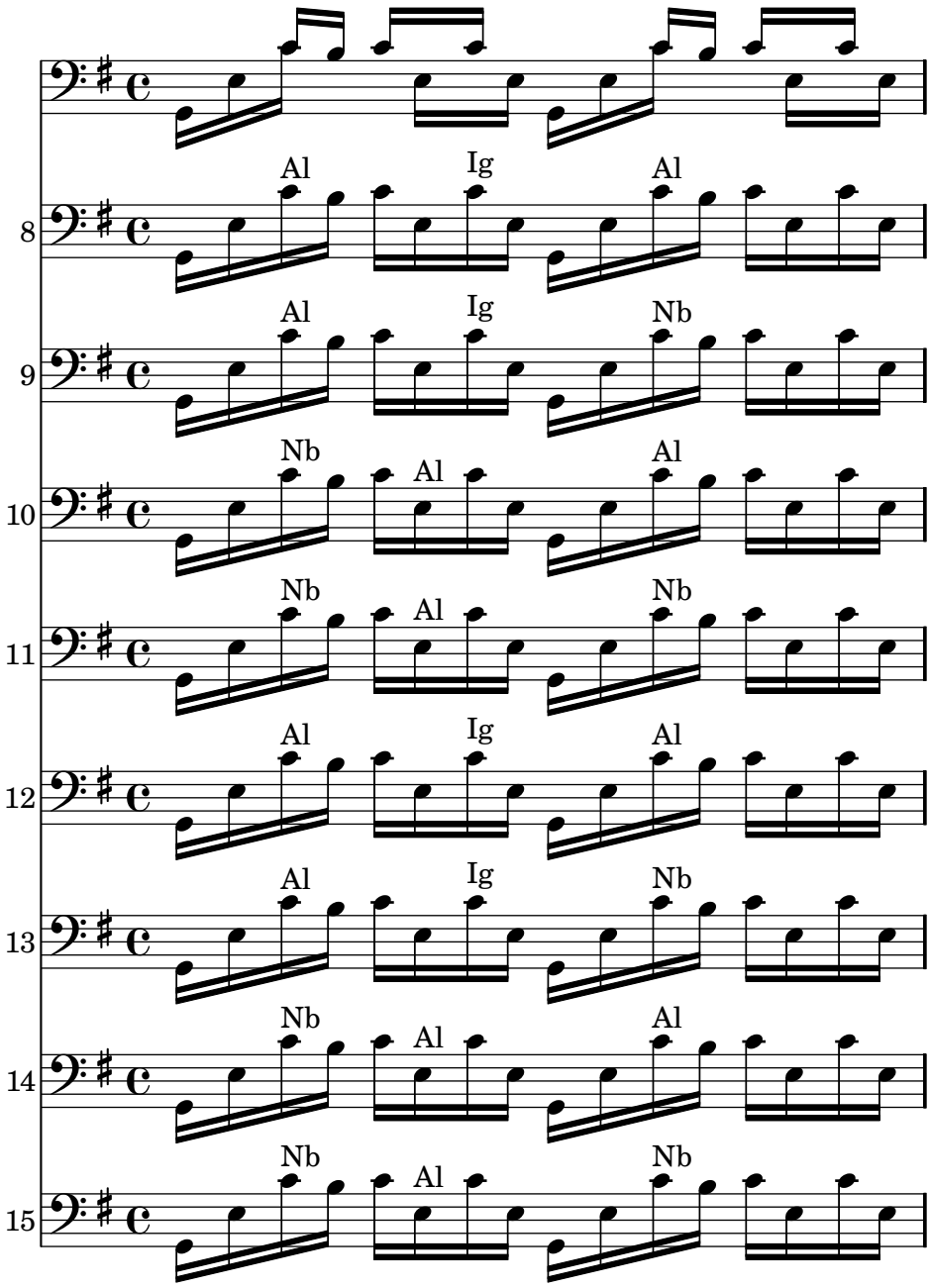
8.3 Depth of Results

Finally, this result is unabridged, but only for the first two bars. Limiting the analysis to only 32 notes, there are still 16 possible readings through these opening measures.


0
Al
Ig
Al
Ig
1
Al
Ig
Al
Ig
2
Al
Ig
Al
Ig
3
Al
Ig
Al
Ig
4
Al
Ig
Nb
Ig
5
Al
Ig
Nb
Ig
6
Al
Ig
Nb
Ig
7
Al
Ig
Nb
Ig


8
Nb
Al
Al
Ig
9
Nb
Al
Al
Ig
10
Nb
Al
Al
Ig
11
Nb
Al
Al
Ig
12
Nb
Al
Nb
Ig
13
Nb
Al
Nb
Ig
14
Nb
Al
Nb
Ig
15
Nb
Al
Nb
Ig


0
Al
Ig
Al
1
Al
Ig
Nb
2
Nb
Al
Al
3
Nb
Al
Nb
4
Al
Ig
Al
5
Al
Ig
Nb
6
Nb
Al
Al
7
Nb
Al
Nb



There is a degree of duplication evident in the results, but this is expected, since the presence of a single small ambiguity creates two slightly divergent readings.

A more sparse data form and presentation would ultimately be desirable, but the presentation as is keeps with the stated aims of maximising information.

The bracketing here is an artefact of lilypond, whereby a bracket that ends on a single note is conglomerated with a second bracket beginning on the same note. This is manifested as longer brackets with the Tag for the second structure above the note that belongs to both the bracketed structures.

Chapter 9

Conclusion and Discussion

A system that can parse musical structure has been created, tested, and presented: the primary goal of the work as presented has been achieved. It has also become apparent that in the intervening years since this work was begun, new avenues for research have emerged for which the work presented here can provide a useful or even crucial component.

This work shows monadic parser combinators that have been modified in a novel manner to allow the production of systematically complete outputs. This is not only the first application of monadic parser combinators in music research, but the modifications made, relating to the parallelisation of results, are also unique and have proven to be an effective way of extending the utility of parsers constructed in this fashion.

System outputs, as presented, are created quickly and objectively, merging analytical and representational tasks in a single process. The choice of lilypond, as a vehicle for presenting music research results in tandem with a score, has

been recently reinforced by its use in systems such as Rameau [32] and PML [15]. Those systems also choose to make this presentation/integration of results and integral part of their corporate system, mirroring the approach taken here.

The depth (or breadth) of results was not required to be reduced at any point within the system (with the caveats set out below regarding representation). This satisfies the supporting statement laid out in the initial declaration, and reinforces the notion that we should not be bound to recreate human processes when employing computation to achieve some research goal.

No *a priori* knowledge of the work analysed was required by the user. While it is not envisaged that an untrained user would use the presented solution with any regularity, removing the ability for an "expert" user to tune a query should in fact increase the utility of the system. If we allow users to delimit the scope of an analysis, we will decrease radically the chances of learning anything new; an expert query will usually seek to verify a hypothesis. A system which merely allows a user to make closed, confirmatory queries more quickly, would be a less beneficial deployment of this technology.

While useful conclusions may be drawn from the results presented in this thesis, it would also be fair to say that the results are not a complete music parsing system. Rather, they demonstrate the effectiveness of the framework which underpins them. That a piece of music can be usefully analysed with such a small set of parsers is pleasing, but it also challenges some preconceptions of how we may understand music, suggesting perhaps that this low-level approach can provide more "information" than was previously expected or assumed.

The results shown previously can be seen to be at a hierarchically low level, which is in marked contrast to the high-level results often presented in academic music analysis. As discussed, it will rarely be practical to perform a manual, low-level analysis of a large work (statistical analysis of cadences in Bach 4-part Chorales being a notable and laborious exception, much aided by such work as Reimenschneider [90]). Aside from this, we might also reflect that as listeners (or performers), our "real time" or in the moment perception of music, as it passes us by, is very much a low-level experience. While we may be able later, either on reflection or as the music progresses, to replay our experience of what we have heard and draw out a more over-arching, high-level understanding, it will never be systematic.

The parsers presented capture only a fraction of musical structure, and more parsers would need to be created, and ideally generated, for a more comprehensive system to be achieved. The formation of the parsers and combinators in this work however represents an effective and workable basis for such a system and it seems that there are no obstacles to structural detection inherent within it. Partially consuming parsers have proved an effective solution to the problem at hand.

One fact which has become apparent is the inadequacy of current score manipulation software to represent multiple levels of structural data. It might even be considered whether a simultaneous graphical representation of all the available structural information is either feasible or desirable, though as noted in Section , a unitary representation is a feature of most extant analyses. It may be considered that currently published scholarly analyses frequently paraphrase the works under analysis in order to convey their findings, but the ability to

abstract parts of a score into parallel structures is absent from all current tools.

Using a score as a source for an analysis has been shown to be both achievable and effective. It is certainly the case that many of the earliest applications of computation in musicology used an (often reduced) representation of the score, which we can ascribe to a lack of raw computational power to deal with the more information rich resources of live or recorded music. The ability to switch from horizontal to vertical representations of the score has been achieved, and is an effective way of surmounting the issues summarised in Section . The option for atomic elements to belong multiple elements as described in Section was shown in the results on page

9.1 Opportunities

Since the inception of this work, a number of fields have emerged within music research wherein the systematic structural information produced by the current system could usefully be employed. These areas include both areas where computation is a fundamental aspect, and also those where computation is used as a tool in some more traditional musicological process.

9.1.1 Performance Markup Language

Pullinger *et al* [79] have drawn attention to the fact that

> *"it has become increasingly obvious that tools available for the presentation and storage of experimental data ... are virtually absent"*

and have made efforts, in their Performance Markup Language (a superset of MusicXML) to allow for musical structure information. PML was presented previously in [15], as a way of integrating both musical and gestural information within a single container format.What is lacking at present is a way to produce this structure information in a quick or even partially automated manner. Producing the information manually is time consuming, as is transforming it into a machine readable format so that it can be imported into the PML. Since the work in this thesis is already interfacing with XML, is automated, and could easily be adapted to output in an alternative computer readable format, it would be eminently suited to plug this gap.

9.1.2 Empirical Musicology

The University of London hosted the first Empirical Musicology conference in April 2008, and this is to be followed up by a second conference at Leeds University in 2010. Groups including the Institute of Musical Research (IMR) at the University of London, the Centre for Performance Science (CPS) at the Royal College of Music, and the Society for Education Music and Psychology Research (SEMPRE), are typical stakeholders.

The empiricism is typically in the digital acquisition of musicological data from performance instances (usually live, but information may still be gathered from an audio recording. MacRitchie *et al* [49], comment that previous systems

> *"...measure variables such as tempo and dynamics, and this is usually done without much consideration of the musical structure."*

but there is also an implication that without a computational approach to generating structural information, the scope for aligning that information is limited.

We can see again that the ability of the system presented in this work to produce structural information, in a short period of time, and in a machine readable format, makes it highly suited to leverage against this emerging field of research.

9.1.3 Musical Instruction Generation

Whilst there is a degree of writing on the subject of structure in performance, the more general texts [21] concern what structures should be brought out in a performance, rather than how that might be achieved. This is certainly understandable since techniques are instrument dependent. Effectively then, we have isolated incidents of how to perform a specific piece on a specific instrument. Of these, journals articles are often enlightening, but difficult to source, while printed editions may be presented without any supporting evidence for the decisions made.

A logical way to proceed could be to use the system presented here to generate structural information about a piece, and compare it with possible instrument executions so that the boundaries coincide; a single performance structure such as a string position, slur or phrasing mark would then reinforce a musical structural. Since the system is automated, and could mark up the results at the same time, experiments could also be created to test hypotheses about the interaction of gestural performance structures and perceived or implied musical structures.

Bibliography

[1] Audiveris, 2008. `http://audiveris.java.dev.net`.

[2] The internet archive. `http://www.archive.org`, 2008.

[3] La mediatheque de l'ircam. `http://mediatheque.ircam.fr`, 2008.

[4] The lester s levy collection of sheet music project. `http://levysheetmusic.mse.jhu.edu`, 2008.

[5] Photoscore. `http://www.neuratron.com`, 2008.

[6] SharpEye music reader. `http://www.recordare.com/sharpeye/`, 2008.

[7] Shazam music portal. `http://www.shazam.com`, 2008.

[8] Variations2 digital music library project. `http://dml.indiana.edu`, 2008.

[9] T W Adorno. On the problem of musical analysis. *Music Analysis*, 1(2):169–187, 1982. trans. M Paddison.

[10] A V Aho, S C Johnson, and J D Ullman. Determenistic parsing of ambiguous grammars. *Communications of the ACM*, 18(8):441–452, 1975.

[11] L Akesson. Functional music. 2008.

[12] J S Bach. *Six Suites pour violoncelle seul.* Francis Salabert S.A., 1929.

[13] J S Bach. *Sechs Suiten fur Violoncello solo.* Barenreiter-Verlag, 1950.

[14] J Backus. Can programming be liberated from the von neumann style?: a functional style and its algebra of programs. *Commun. ACM*, 21(8):613–641, 1978.

[15] S Pullinger D McGilvray N Bailey. Music and gesture: Performance visualisation, analysis, storage and exchange. In *Proceedings of the International Computer Music Conference*, 2008.

[16] D Bainbridge, G Bernbom, M Wallace, A P Dillon, M Dovey, J W Dunn, M Fingerhut, I Fujinaga, and E J Isaacson. Digital music libraries - research and development. In *JCDL '01: Proceedings of the 1st ACM/IEEE-CS joint conference on Digital libraries*, pages 446–448. ACM Press, 2001.

[17] N K Baker and T Christensen, editors. *Asethetics and the Art of Composition in the German Enightenment.* Cambridge Studies In Music Theory and Analysis. Cambridge University Press, 1995.

[18] H Barendregt. The impact of the lambda calculus in logic and computer science. *The Bulletin of Symbolic Logic*, 3(2):181–215, June 1997.

[19] A H Benade. *Fundamentals of Musical Acoustics.* Dove Publications Inc., 1990.

[20] I Bent and W Drabkin. *Analysis.* The New Grove Handbooks In Music. Macmillan, 1986.

[21] W Berry. *Musical Structure and Performance.* Yale University Press, 1989.

[22] S G Blackburn. Search by humming. Technical report, University of Southampton, 1997.

[23] D Blostein and H S Baird. A critical survey of music image analysis. In Baird, Bunke, and Yamamoto, editors, *Structured Document Image Analysis*. Springer-Verlag, 1992.

[24] R Bod. Probabilistic grammars for music. In *Proceedings of Belgian-Dutch Conference on Artificial Intelligence*, 2001.

[25] J S Bolton. *Gestural Extraction from Musical Audio Signals*. PhD thesis, University of Glasgow, 2005.

[26] H M Brown. *Embellishing Sixteenth-Century Music*. Number 1 in Early Music Series. Oxford University Press, 1976.

[27] J Burmeister. *Musical Poetics*. Music Theory Translation Series. Yale University Press, 1993. trans. B V Rivera.

[28] M Cahill. The translation of finale's enigma file format for cpnview. Technical report, University of Limerick, 1998.

[29] E Cambouropoulos. *Towards a General Computational Theory of Musical Structure*. PhD thesis, The University of Edinburgh, 1998.

[30] E Chew. The spiral array: An algorithm for determining key boundaries. In *Lecture Notes in Artifical Inteligence 2445*, pages 18–31. Springer-Verlag, 2002.

[31] A Church. *The Calculi of Lambda-Conversion*. Number 6 in Anals of Mathematics Studies. Princeton University Press, 1941.

[32] A T Passos M S Sampaio P R Kroger G Cidra. Functional harmonic analysis and computational musicology in rameau. In *Proceedings of the XII Brazilian Symposium on Computer Music*, 2009.

[33] E Clark, R Parncutt, M Raekallio, and J Sloboda. Talking fingers: an interview study of pianists' views on fingering. *Musicae Scientae*, 1(1):87–107, 1997.

[34] M Clynes. What can a musician learn about music performance from newly discovered microstructure principles? In A Gabrielsson, editor, *Action and Perception in Rhythm and Music: Third International Conference on Event Perception and Action*, number 55, pages 201–233. Royal Swedish Academy of Music, 1987.

[35] M R Cohen and I E Drabkin, editors. *A Source Book in Greek Science*. Harvard University Press, 1948.

[36] N Cook. *A Guide To Musical Analysis*. J M Dent & Sons Ltd, 1987.

[37] D Cope. *Computers and Musical Style*. Computer Music and Digital Audio Series. Oxford University Press, 1991.

[38] D Cope. *Virtual Music : Computer Synthesis of Musical Style*. MIT Press, 2001.

[39] C Cronin. Repp v. webber. `http://www.ccnmtl.columbia.edu/projects/law/library/cases/case_reppwebber.html`.

[40] K Czerny. *School of practical composition*. R Cocks & Co., 1849. Translated by J Bishop.

[41] R B Dannenberg. The canon score language. *Computer Music Journal*, 13(1):47–56, 1989.

[42] R B Dannenberg, P McAvinney, and D Rubine. Arctic: A functional language for real-time systems. *Computer Music Journal*, 10(4):67–78, 1986.

[43] J J de Momigny. *Cours Complet d'Harmonie et de Composition*. Paris, 1806.

[44] F de Saussure. *Course In General Linguistics*. Fontana, 1974.

[45] D Deutsch, editor. *The Psychology of Music*. Academic Press, 1999.

[46] S di Ganassi dal Fontego. *Opera Intitulata Fontegara: a treatise on the art of playing the recorder and of free ornamentation*. Berlin-Lichterfelde, 1959. trans. P Hildemare.

[47] G Diruta. *Il Transilvano*. Bibliotheca Organologica. Frits Knuf Buren, 1983. Facsimile edition with introduction by E J Soehnlen and M C Bradshaw.

[48] A Forte and S E Gilbert. *Introduction to Schenkerian Analysis*. W W Norton, 1982.

[49] J MacRitchie N Bailey G Hair. Multi-modal acquisition of performance parameters for analysis of chopin's b flat minor piano sonata finale op 35. Technical report, Digital Music Research Network Workshop, Queen Mary University, London, 2006.

[50] H Helmholtz. *On The Sensations of Tone As A Physiological Basis For The Theory of Music*. Dover Publications Inc, 1954.

[51] W B Hewlett. A base-40 number-line representation of musical pitch notation. *Musikometrica*, 4, 1992.

[52] W B Hewlett and E Selfridge-Field. Base-40 arithmetic for notation-based applications. `http://esf.ccarh.org/MusicTheory_Tutorials/Base40_abstract03.pdf`.

[53] H Hild, J Feulner, and W Menzel. Harmonet: A neural net for harmonizing chorals in the style of j.s. bach. *Advances in Neural Information Processing*, 4:267–274, 1992.

[54] L A Hiller and L M Isaacson. *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill, 1959.

[55] P Hudak. Conception, evolution, and application of functional programming languages. *ACM Computing Surveys*, 21(3):359–411, 1989.

[56] P Hudak. Haskore music tutorial, 2000.

[57] P Hudak, T Makucevich, S Gadde, and B Whong. Haskore music notation – an algebra of music. *Journal of Functional Programming*, 6(3), June 1996.

[58] P Hudak and P Wadler. Report on the functional programming language haskell. Technical report, Yale University, November 1989.

[59] J Hughes. Why Functional Programming Matters. *Computer Journal*, 32(2):98–107, 1989.

[60] G Hutton. Higher order functions for parsing. *Journal of Functional Programming*, 2(3):323–343, July 1992.

[61] G Hutton and E Meijer. Monadic parser combinators. Technical Report NOTTCS-TR-96-4, University of Nottingham, 1996.

[62] J Jeuring and D Swierstra. Grammars and parsing. `http://www.cs.uu.nl/people/johanj/publications/MAIN.pdf`, 2001.

[63] M P Jones et al. Haskell users gofer system. `http://www.haskell.org/hugs`.

[64] S Peyton Jones. Special issue: The journals of functional programming. haskell 98. *Journal of Functional Programming*, 13(1):1–255, January 2003.

[65] O Jorgensen. *Tuning the Historical Temperaments by Ear*. The Northern Michigan University Press, 1977.

[66] H Keller. A slip of mozart's: Its analytic significance. *Tempo*, 42(winter):12–15, 1956-57.

[67] H Keller. Functional analysis: its pure application. *The Music Review*, 18:202–206, 1957.

[68] J P Kirnberger. *Die Kunst des reinen Satzes in der Musik aus sicheren Grundsätzen hergeleitet und mit deutlichen Beyspielen erläutert*. G J Decker and G L Hartung, 1774-79.

[69] H C Koch. *Versuch einer Anleitung zur Composition*. A D Böhme, 1882-93.

[70] K Lemström. *String Matching Techniques for Music Retrieval*. PhD thesis, Department of Computer Science, University of Helsinki, 2000.

[71] K Lemström and V Makinen. On minimizing pattern splitting in multi-track string matching.

[72] K Lemström and V Mäkinen. On finding minimum splitting of pattern in multi-track string matching. Symposium on Combinatorial Pattern Matching, 2003.

[73] K Lemström and J Tarhio. Searching monophonic patterns within polyphonic sources. Content-Based Multimedia Information Access (RIAO), 2000.

[74] F Lerdahl and R Jackendoff. *A Generative Theory of Tonal Music*. MIT Press, 1983.

[75] N Listenius. *Musica*. Colorado College Music Press, 1975. Translated by A Seay.

[76] J Locke. *An Essay Concerning Human Understanding*. Everyman, 1993.

[77] K MacMillan, M Droettboom, and I Fujinaga. Gamera: Optical music recognition in a new shell. International Computer Music Conference, 2002.

[78] S Marlow, S Peyton Jones, et al. Glasgow haskell compiler. `http://www.haskell.org/ghc`.

[79] S Pullinger N Bailey J MacRitchie M McAllister. Computer assisted analysis and display of musical and performance data. In *Proceedings of the International Symposium on Performance and Science*, 2009.

[80] J McCarthy. Recursive functions of symbolic expressions and their computation by machine. Technical report, 1959.

[81] J McCarthy. History of programming languages. In *Proceedings of the ACM SIGPLAN Conference*, pages 173–197, 1978.

[82] R A McIntyre. Bach in a box: The evolution of four-part baroque harmony using the genetic algorithm. *Proceedings of the IEEE Conference on Evolutionary Computation*, 14(3), 1994.

[83] D Meredith, G A Wiggins, and K Lemström. Pattern discovery and pattern matching in polyphonic music and other multidimensional datasets. In *Music, Minds, Machines Seminar*. Faculty of Music, University of Edinburgh, June 2001.

[84] R Middleton. *Studying Popular Music*. Open University Press, 1990.

[85] E Moggi. Computational lambda-calculus. Technical Report ECS-LFCS-88-66, Laboratory for Foundations of Computer Science, 1989.

[86] J-J Nattiez. *Music and Discourse: Towards a Semiology of Music*. Princeton University Press, 1990. Translated by Carolyn Abbate.

[87] R Parncutt, J S Sloboda, E F Clarke, M Raekallio, and P Desain. An ergonomic model of keyboard fingering for melodic fragments. *Music Perception*, 14(4):341–382, 1997.

[88] C S Peirce. *The Essential Peirce: Selected Phiolosophical writings*. Indiana University Press, 1992.

[89] J-P Rameau. *Treatise on Harmony*. Dover, 1971.

[90] A Riemenschneider, editor. *371 Harmonized Chorales and 69 Chorale Melodies with Figured Bass*. G Schirmer, 1941.

[91] J Riepel. *Anfangsgründe zur musicalischen Setzkunst*. J L Montag, 1754.

[92] J Riepel. *Grundregeln zur Tonordnung insgemein*. U Wagner, 1755. Part II of Anfangsgründe.

[93] J Riepel. *Erläuterung der betrüglichen Tonordnung*. J J Lotter, 1765. Part IV of Anfangsgründe.

[94] N Ruwet. Methodes d'analyse en musicologie. *Revue Belgique*, 20:65–90, 1966.

[95] N Ruwet. Methods of anlysis in musicology. *Music Analysis*, 6(1–2):11–36, 1987.

[96] N Röjemo et al. Nearly a haskell compiler. `http://www.haskell.org/nhc`.

[97] S Sadie and J Tyrrell, editors. *The New Grove Dictionary of Music and Musicians*. Macmillan, 2001.

[98] S I Sayegh. Fingering for string instruments with the optimum path paradigm. *Computer Music Journal*, 13(3):76–84, 1989.

[99] A Schoenberg. *Theory of Harmony*. Univeristy of California Press, 1993.

[100] A Schoenberg. *Coherence, Counterpoint, Instrumentation, Instruction in Form/Zusammenhang, Kontrapunkt, Instrumentation, Formenlehre*. University of Nebraska Press, 1994.

[101] E Selfridge-Field. Conceptual and representational issues in melodic comparison. *Computing in Musicology*, 11:3–64, 1998.

[102] J Skibinski. Collection of haskell modules. http://web.archive.org/web/20010716012429/http://www.numeric-quest.com/haskell/index.html, july 2001.

[103] S W Smoliar. A computer aid for schenkerian analysis. *Computer Music Journal*, 4(2):41–59, 1980.

[104] D Strahan. Beethoven: The illuminati & the recycled themes. `http://www.revolve.com.au/polemic/beetintro.html`.

[105] E Tarasti. *Myth and Music*. Approaches to Semiotics. Mouton, 1979.

[106] S Thompson. *The Craft of Functional Programming*. Addison-Wesley, 1999.

[107] Aristotle trans. E S Forster. *Problems*, volume 19. Oxford University Press, 1927.

[108] G Visconti. *Concerto in F Major*. Number HH014.FSC. Edition HH.

[109] P Wadler. Comprehending monads. In *Conference on LISP and Functional Programming*, pages 61–78. ACM, 1990.

[110] J Ziv and A Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

Appendix A

Additional Haskell Code

A.1 Additional functions for Chapter 5

```
always :: a → Parser a
always a = λinp → [(a, inp)]
```

```
never :: Parser a
never = λinp → []
```

```
duplet :: Parser Seq
duplet =chorditem `bind` λx →
        chordscan `bind` λy →
        always (x : [y])
```

```
chorditem :: Parser Crd
chorditem = λ inp → case inp of
  [] → []
  (x : xs) → [(x, xs)]


chordscan :: Parser Crd
chordscan = λ inp → case inp of
  [] → []
  (x : xs) → [(x, x : xs)]


notestepup :: B40 → B40 → Bool
notestepup n1 n2 = case ((forty2Seven n1) `b7Interval` (forty2Seven n2)) of
  1           →True
  otherwise →False


notestepdown :: B40 → B40 → Bool
notestepdown n1 n2 = notestepup n2 n1



chordstepdown :: Crd → Crd → Bool
chordstepdown c1 c2 = cMapAny c1 c2 notestepdown



seqstepdown :: Seq -> Bool
seqstepdown [] = True
seqstepdown s@(c : cs)
```

$$
\begin{aligned}
&\mid (chordstepdown\ c\ (head\ cs))\ ==\ True = (seqstepup\ (tail\ cs))\\
&\mid otherwise \qquad\qquad\qquad\qquad\qquad\qquad = False
\end{aligned}
$$

$$
\begin{aligned}
&bind\ ::\ Parser\ a\ \rightarrow\ (a\ \rightarrow\ Parser\ b)\ \rightarrow\ Parser\ b\\
&p\ `bind`\ f\ =\ \lambda\ inp\ \rightarrow\ concat\ [f\ v\ inp' \mid (v, inp')\ \leftarrow\ p\ inp]
\end{aligned}
$$

A.2 Additional functions for Chapter 6

$$
\begin{aligned}
&tripletnote :: Parser\ Seq\\
&tripletnote = noteitem\ `bind`\ \lambda\ x\ \rightarrow\\
&\qquad\qquad noteitem\ `bind`\ \lambda\ y\ \rightarrow\\
&\qquad\qquad notescan\ `bind`\ \lambda\ z\ \rightarrow\\
&\qquad\qquad always\ ([x] : [y] : [[z]])
\end{aligned}
$$

$$
\begin{aligned}
&pedalBin\ seq \mid (length\ seq)\ >\ 2\ =\ chordpedal\ (seq!!0)\ (seq!!2)\\
&\qquad\qquad \mid otherwise\ =\ False
\end{aligned}
$$

$$
\begin{aligned}
&chordpedal\ c1\ c2 \qquad\qquad = cMapAny\ c1\ c2\\
&notepedal\ notepedal\ n1\ n2 = n1\ `nexact`\ n2
\end{aligned}
$$

A.3 Additional functions for Chapter 7

$$
\begin{aligned}
&mlex :: [(Parser\ Seq,\ String)]\ \rightarrow\ Parser\ [Token]\\
&mlex = many.(foldr\ op\ never)
\end{aligned}
$$

$$many \quad :: Parser\ a\ \rightarrow\ Parser\ [a]$$
$$many\ p = (some\ p)\ `choice`\ always\ []$$

Appendix B

Computing Environment

The initial research fot this work was coompleted on a computing plaform consiting of an Intel Pentium II 233, 192MB RAM, running Debian GNU/Linux Testing release. All typesetting was undertaken using LyX, BibTeX, LaTeX and lilypond-book.

Software versions at time of first submission:

GNU Lilypond version 2.4.5

Denemo version 0.7.3 beta 2

GHC version 6.4

HaXml version 1.13

Software versions at time of final submission:

GNU Lilypond version 2.10.33

Denemo version 0.7.7

GHC version 6.8.2